1-2005

Web Service infrastructure for supply chain

Pradip R. Chandratre

Web Service Infrastructure for Supply Chain

Pradip R. Chandratre

B. Tech. (Chemical Technology)
University of Mumbai, Institute of Chemical Technology

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Industrial and Systems Engineering in the Kate Gleason College of Engineering of the Rochester Institute of Technology

December 2005

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MASTER OF SCIENCE DEGREE THESIS

The M.S. Degree thesis of Pradip R. Chandratre has been examined and approved by the thesis committee as satisfactory for the thesis requirement for the Master of Science degree.

Sudhakar Paidy

Dr. Sudhakar Paidy, Ph. D. Advisor

Moises Sudit

Dr. Moises Sudit, Ph. D.

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: WEB SERVICES INFRASTRUCTURE FOR SUPPLY CHAIN

Name of author: PRADIP R CHANDRATRE
Degree: MASTER OF SCIENCE
Program: INDUSTRIAL AND SYSTEMS ENGINEERING
College: COLLEGE OF ENGINEERING

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, PRADIP CHANDRATRE, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: P. R. Chandratre Date: 12/14/2005

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, PRADIP CHANDRATRE, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: P. R. Chandratre Date: 12/14/2005

Dedicated to Aai, Aaba, Prachi and Dipti didi.

Acknowledgement

Lot of effort was put into completing this thesis document, by me and many others. I want to thank my advisor Dr. Paidy for introducing the topic of Web Services to me and guiding me along the way while researching for this thesis. Dr. Paidy has been the most instrumental in making sure that this thesis makes a significant contribution to the academic literature. I also want to thank Dr. Sudit for his helpful critique and encouragement.

I want to thank my fellow graduate student Mr. Bharani Govindasamy for pointing out many relevant research papers as well as for offering help while doing this research work. Many thanks are due to the computer science department, especially Dr. Schreiner, under whom I learnt the skills required for putting together an extensive .NET application, such as the one developed for this thesis.

I also want to thank Mr. Adam Goodenough and Ms Buehler from the Publishing and Scholarship Support Services at the Wallace Library for editing this thesis document.

I want to thank each and every person who has affected my work directly or indirectly and made this thesis's success possible.

Abstract

Managing a supply chain is one of the most complicated tasks today when erratic changes in demand must be met as soon as possible for staying competitive, while dealing with multitude of business partners that are involved in the chain. It is imperative that any changes in a corporation's product/service demand be immediately communicated with its suppliers and logistic service providers. This task of communication has long been overtaken by computerized systems from the telephones and fax machines. The computer technologies being used so far to connect two businesses are proving to be too rigid in today's world of mergers, acquisitions, new business deals etc which bring in the task of tying the disparate computer systems of these different organizations. To solve this problem, the enterprise software industry has developed new standards and a new design for constructing inter-organization applications, collectively known as the Web Service technology.

This paper demonstrates how this technology works and how it can be applied to the problem of supply chain management. This paper describes the principles of Web Services and its features like UDDI. A demonstrative supply chain infrastructure is created using the Web Service technology which shows the ease of creating new communication links with new supply chain partners without having to invest in costly computer technology resources. The paper will show that the adoption of Web Services and adoption of standard business language OAGIS will make the task of supply chain communication as easy as plug and play.

Table of Contents

List of Figures

List of Tables

1. Introduction

Supply chain management goes to the heart of any organization's performance and success in its industry. As an industrial engineer, it is important to recognize the factors affecting the performance of a supply chain and attempt to improve it. Managing an efficient supply chain leads to benefits like reduced inventory costs and short response time to changes in demand which, in turn, leads to an increase in the bottom line for the businesses involved. An important factor in making the supply chain efficient is how easy it is for a supplier to be able to select from a wide variety of available logistics providers. It will be beneficial to the supplier, in term of money saved and good will earned from the customer, to be able to select a carrier who will deliver the goods at an optimal combination of a cheap rate and a quick delivery time. A similar argument can be made if the manufacturer is to select the carrier. Also, the supply chain can be at its maximum flexibility if a manufacturer can select its supplier according to its own needs. A flexible supply chain will allow a manufacturer to select a supplier from the widest list of choices, deliver the exact quantity of goods that it needs at the best rate possible and as close as possible to the date when the goods are needed.

This is a utopian view of supply chain architecture that obviously does not exist in practice today. There can be many reasons for that. To make this vision possible, a manufacturer will have to get into legal contracts with as many suppliers as it can so that it can make use of their services on demand. The legal details of such situations are beyond the scope of industrial engineering, but it can be safe for us to assume that it is possible to form legal contracts that will allow businesses to get into an "on-demand" style of interaction. This style of interaction results in one business being able to ask for another business's services with no strings attached. For example, if a manufacturer asks for a quote from multiple suppliers and it likes what one of them is offering, the manufacturer should be able to conduct business with it just as easily as anyone else, even though they may never have conducted business before (and may never do so again).

Even though this style of interaction is possible in principle, there are plenty of problems when we get down to the low level implementation, as will be illustrated in the following discussion. When a manufacturer has selected a supplier, there are plenty of transactions that still need to be done. These transactions involve processing a purchase order, sending/receiving delivery status information/confirmation, processing an invoice, and many other transactions that are possible. Conducting such transactions requires that there should be a reliable means of communication between the two parties. Communication using phone, fax machines and email is extremely unwieldy for complex supply chains. The alternative is to use network technologies, such as CORBA, EDI, RMI, etc. which will form the communication links between the two supply chain partners.

The usage of network technologies solves the problem of communication when the customer has decided on a supplier. The suppliers and customers will use one of the technologies above, build their networks accordingly and start communicating. In the process they will have spent a significant amount of resources, including, time and money, to get this network up and running. The next time the customer wants some

goods and selects an entirely different supplier, it will possibly have to change the entire network infrastructure to adopt the same technology that the new supplier is using or force the new supplier to change its infrastructure, neither of which may be very feasible. Hence, to avoid such troubles, the manufacturer may instead decide to go with the supplier it already was doing business with. This runs contrary to the concept of on-demand business, as the lack of technical communication flexibility prevents organizations from freely forming business alliances with different business partners. The root cause of this is the lack of standardization.

If this problem is to be solved, we need a technology that everybody can agree on. It is possible to agree on, say, XYZ vendor's product as the standard technology, because it has been the widest used so far. But it would be a proprietary technology developed by a single organization or by a small number of vendors. Accepting this technology as a standard may lead to problems in the future, such as fees being required to use the technology, lack of transparency in how the technology has been developed and how it works and an inability for the supply chain industry to contribute to the improvement of the technology. This applies to any proprietary technology. At the same time, an open technology alone is not sufficient. It has to be backed by major software vendors so that business organizations can feel secure about using it, believing that it will be easy to get technical support for these technologies.

The newly emerging Web Service standard promises to solve this problem by mixing the best of both worlds. Web Services is an open technology to which many software vendors as well as end user business organizations are contributing. That means that the technology will evolve openly and freely as it matures. At the same time, this technology is backed by major software vendors, assuring that it will continue to receive technical support and will not diminish.

Web Services technology, being a very flexible technology, does not put any restrictions on what purposes it is used for. This technology has the great promise of providing an on-demand infrastructure for supply chain architecture, but before it can be used there are some issues that need to be sorted out. Web Services allow any kind of message to be communicated. It is necessary that these messages and their formats be standardized; otherwise, if each manufacturer comes up with its own formulation of how these messages should be structured, we will be back to the problem of a lack of standardization. Hence, the semantic structure of supply chain messages must be standardized and agreed upon. Even after this, there are some details that need to be agreed upon so that a supply chain with free partner-swapping can be constructed.

This thesis proposes an architecture that takes all of these issues into account and creates a supply chain network in which it is possible to add any number of suppliers, customers and logistics providers, form on-demand links with any other business partners and carry out supply chain transactions. The following chapters of this document will examine and elaborate each of the facts mentioned above and each of the claims made.

Chapter 2 summarizes what sort of research has already been done in this arena and what this thesis will contribute. Chapters 3 to 5 explain Web Services and the technical details, as well as provide a comparison with earlier network communication

technologies and present some case studies of Web Services illustrative of different ways in which the services can be used.

Chapter 6 gives some background on typical supply chain participants and shows some cases of real-life supply chains. Chapter 7 shows the need for a semantic communication standard for the messages exchanged between two business partners and explains the widely used semantic standards.

In Chapter 8, the actual proposed architecture is described. The chapter also explains how the architecture can be implemented using the Microsoft .NET. Finally, chapter 9 presents the conclusions reached in this thesis.

2. Literature Review

How successfully an organization is able to survive and prosper in its business environment depends in a large part on how quickly, efficiently and extensively it is able to create communication networks with its business partners. Trends of success and failure within businesses indicate that forming networks with a company's business partners is vital for the success of the company (Hengst 2001).

As soon as the use of computers for data storage and processing became prevalent in business, technologies like RPC, EDI, CORBA, and Java RMI were created and used for inter-organization communication via two remote computer systems. Each one of these technologies has advantages over the earlier one. In RPC (Remote Procedure Call), the complexities of network communication are hidden from the programmer and requesting a service from another computer located on a network is made much more transparent (Vondrak, 1997). EDI (Electronic Data Interchange) is primarily used for the interchange of documents like purchase orders between businesses (Jilovec, 1998). CORBA is a highly successful (Ryan, 2000) communication architecture created by the Open Management Group (OMG 2005) which has been used extensively for the purpose of remote computer system communication. All of these technologies will be discussed in more detail later in this document.

These technologies, though useful in certain scenarios, have limitations of one kind or another (Stal 2002). The single disadvantage common to all of these technologies is a lack of standardization when it comes to low level implementation. Since one implementation of the technology is almost always incompatible with other implementations, adoption of one of these technologies requires other communicating business entities to adopt the same implementation. This results in locking in resources and creating tight coupling between the two business entities. Additionally, these technologies rely on proprietary software and lack widely-reliable network infrastructure. These and other such disadvantages for RPC, CORBA, and RMI have been documented by Zahavi (1999), Greenfield (2003), and Baclawski (1998), respectively.

Development of Web Service technology by the World Wide Web Consortium started in 2002 and kept in mind the lessons learned by the limitations of earlier technologies (W3C 2005). Web Service technology is being developed not by a single company, but by a consortium of organizations which include rivals like Microsoft and Sun working together along with other software companies. This gives the Web Services standards wide acceptability and assures users that these standards will be honored by the major players of the software world. Major IT corporations are supporting Web Services through their products, such as Microsoft's .NET and Sun Microsystems's J2EE. Some commercial services, such as HP's *Adaptive Enterprises* and IBM's *On-Demand E-Business*, are using Web Services as the underlying technology.

Web Service technology, and the Service Oriented Architecture (SOA) that Web Services enable, have the potential to solve many of the problems that plagued earlier technologies (Rhody 2002). The previous approaches failed to address key business network issues like interoperability, scalability, heterogeneity and decoupling of

information systems, which are issues that Web Services have been designed to handle (Msanjila 2005). According to Hongbing (2004), Web Services hold the potential to simplify business processes and Papazoglou (2003) illustrates the usability of Web Services for facilitating business processes by addressing the issues of coordination, monitoring, conformance and quality of service composition. Web Services can be used to enable two applications on different platforms to exchange information from databases and expose internal applications to the internet (Kreger 2003). Hansen et al (2002) investigate how Web Services can be used to extract and integrate business data from two incompatible computer systems. Kabassi and Virvou demonstrate the use of Web Services in the context of web-based learning using a multi-agent architecture. Additionally, Web Service researchers have focused on standardizing protocols, languages, models, etc., for the proper use of this technology (Msanjila 2005).

Supply chain management requires a lot of agility and flexibility and can benefit from Web Services technology (Burt 2003). Paik et al (2004) describe an implementation of Web Services in the area of supply chain management which uses IBM's BPEL4WS (Business Process Execution Language for Web Services) to describe the business process. Hassan (2005) as well as Xu et al (2004) propose a supply chain architecture using Web Service intelligent agents. These research articles clearly signify the additional value that can be achieved by incorporating Web Services into supply chain management, but are not clear on how the Web Services in question exchange information or how to structure the content.

Although Web Services make communication protocols standard in terms of Simple Object Access Protocol (SOAP) messages, they do not specify payload construction details. If true loose coupling and on the fly integration between any two business entities is to be achieved, it is necessary to have a consensus on the construction of the payload as well. De facto, this payload is going to be an XML document, but the tree structure of the document should be known to all concerned computing systems. For example, if a manufacturer is sending purchase information to the supplier, the payload should standardize the location of the quantity information, the unit of measure, etc.

This applied research study is an attempt at developing an information system architecture for a supply chain that makes use of the loose coupling capability of Web Services and a vital semantic OAGIS standard. The Open Applications Group is a non-profit group which builds XML standards for B2B (Business to Business) integration (OAGi 2005). This group has devised an Integration Specification known as OAGIS which is a collection of XML schemas that can be used as a standardized language of communication. This work will also use the Web Service registry standard UDDI (Universal Description, Discovery and Integration) which has not been widely adopted in much of the research so far. The thesis work will test the ease of coupling between two business entities and the ease of adding a new business entity with a specific role (e.g. Supplier, Consumer, Service Provider, etc.) to the supply chain of an organization. The usefulness of the architecture is demonstrated and measured by the ease, cost, effort and timeliness with which these additional entities can be added to the supply chain.

References:

Baclawski, Ken. 1998. Java RMI Tutorial. Retrieved January 18, 2005 from http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

Burt, D.N., D.W. Dobler, and S.L. Starling 2003. *World Class Supply Management: The Key to Supply Chain Management*, 7th ed. McGraw-Hill.

C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, and D. Levine. May 2000. "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0", in *Proceedings of the* 6th *IEEE Real-Time Technology and Applications Symposium*, IEEE, Washington DC.

Greenfield, P. March 2003. Web Services – Facts, Fables and Future. Retrieved January 11, 2005 from http://ict.csiro.au/MU/Trends/Presentations/CSIROWebServices.pdf

Hansen, M., Madnick, S., and Siegel, M. 2002. "Data Integration Using Web Services", Working paper 4406-02, MIT Sloan School of Management.

Hassan, U, Ben Soh. 2005. "Web Service Intelligent Agent Structuring for Supply Chain Management (SCM)", *eee*, vol. 00, pp. 329-332.

Hengst, M., Sol, H.G. 2001. "The impact of information and communication technology on interorganizational coordination", In Proceedings: *Hawaii International Conference in System Sciences*, Island of Maui, Hawaii.

Hongbing Wang, Joshua Zhexue Huangc, et al. 2004. "Web services: problems and future directions", *Journal of Web Semantics*, vol. 1, no. 3.

Jilovec, Nahid. 1998. The A to Z of EDI, Duke Press, p xv.

Kabassi, K., and Virvou, M. 2003. "Using Web Services for Personalised Web-based Learning", Educational Technology & Society, vol. 6, no. 3, pp. 61-71.

Kreger, H. 2003. "Fulfilling the Web Services promise", *Communications of the ACM*; vol. 46, no. 6.

Msanjila, S., Tewoldeberhan, T.W., Bockstael-Block, W., Janssen, M., Verbraeck, A. 2005. "E-supply chain orchestration using Web Service technologies: A case using BPEL4WS", In *Proceedings of IRMA 2005 Conference*, San Diego, US.

OAGi 2005. "About [OAGi]", http://openapplications.org/global/intro.htm.

OMG 2005. "Open Management Group", http://omg.org.

Paik, I, M.S. Hossain and M.T.Hossain. 2004. "Automating Business Transaction and Negotiation in Supply Chains Using Web Service Composition". Computer and Information Technology, vol. 5, no. 4 pp. 248-253.

Papazoglou, M.P, Georgakopoulus, D. 2003. "Service-Oriented Computing", *Communications of the ACM*, vol. 46, no. 10.

Rhody, S. 2002. "Why Web Services", *Web services Journal*, vol. 2, no. 5.

Stal, M. 2002. "Web services: beyond component-based computing", *Communications of the ACM*, vol. 45, no. 10.

Vondrak, Cory. June 1997. Remote Procedure Call, Retrieved January 18, 2005 from http://www.sei.cmu.edu/str/descriptions/rpc.html

W3C. 2005. "Web Services Activity: History", Retrieved 2005 from http://www.w3.org/2002/ws/history.html

Wainewright, P. (2000 January), "Anatomy of an ASP: Computing's New Genus", *ASP News Rev.*

Xu, Q, Qiu, R.G., and Fuyuan Xu (2004 October), "Integration of Web services and agents for supply chain system collaboration", *Systems, Man and Cybernetics, 2004 IEEE International Conference on.*

Zahavi, Ron. (1999). Enterprise Application Integration with CORBA, Wiley Computer Publishing, pg. xxxix.

3. Web Services

Introduction to Web Services

Web Services are software programs which have the capacity to interact with any other software programs, regardless of the language in which the two programs are written, as long as the two software programs adopt the same standards of communication. The idea of Web Services is to allow communication between two computer programs without any human intervention. Web Services are self contained, self-describing, modular applications that can be published, located and invoked across the Web (IBM Glossary).

The currently accepted standards are XML communication in SOAP - Simple Object Access Protocol. So it can be said that if XML is the language in which the two programs communicate, then SOAP is the grammar.

According to Businessweek.com, "A Web Service is a software application available over a network – usually the internet – that uses a standardized Extensible Markup Language (XML) messaging system and is not tied to any operating system or programming language."

W3C describes Web Services as the programmatic interfaces made available for application-to-application communication (W3C, par. 1).

The motivations behind the Web Services are more from the Business point of view than from pure technical point of view. Web Services enable loose coupling between a service provider and a service requestor. So in an ideal Web Services world, a service requestor can change his/her service provider without much of investment if he/she does not like the service provider's quality of service. Before the era of Web Services, doing something like this was very expensive, because it was necessary for the service requestor to invest in an entire software infrastructure that would support the software used by the service provider. This is called tight coupling between applications. But Web Services hide the inner software infrastructure from the outside world and use the standard XML over SOAP for communication. Hence, there is no compulsion to use any specific software. Also the modification in one of the party's software infrastructure does not have any impact on the communication.

Within a single enterprise itself, instead of having one big enterprise-wide software, different components built by different vendors could be mixed and matched to create an enterprise application unique to the enterprise without any of the development work (Gralla). Web Services modularize business processes and, hence, become easy to understand for businesspersons. Each Web Service represents a particular function. A service focuses first on what is being done, not how. By thinking in terms of services, the focus is on the business function, not the underlying technology (Connell, par. 17).

Web Services can be used as an extension to our existing applications. For example, if we have a computer program which writes its output to a database, we can develop a

Web Service which converts the database information into XML format and present it to the outside world.

So in summary, Web Services can do the following (Shah, par. 11):

- Interact between services on any platform, written in any language.
- Conceptualize application functions into task, leading to task-oriented development and workflows.
- Allow for loose coupling, which means that interactions between service applications may not break each time there is a change in how one or more services are designed or implemented.
- Adapt existing applications to changing business conditions and customer needs.
- Provide existing or legacy software applications with service interfaces without changing the original applications, allowing them to fully operate in the service environment.
- Introduce other administrative or operations management functions, such as reliability, accountability, security, etc, independent of the original function, thus increasing its versatility and usefulness in the business computing environment.

Web Services Technology

The Web Service software can be written in any language that is suitable. This software has to be able to communicate with other software applications. XML has become a de facto format for representing data for exchange between software components and agents (DevelopMentor 6). XML (eXtensible Markup Language) is a simplified metalanguage derived from the Standard General Markup Language (Uanny 76). XML is used to 'mark-up' data (i.e. to give a structure to the data), so that it can be readily recognized by a computer program. The Web Services use XML data as inputs and outputs.

The IBM's idea of the Web Services conceptual stack is shown in the figure 3-1:

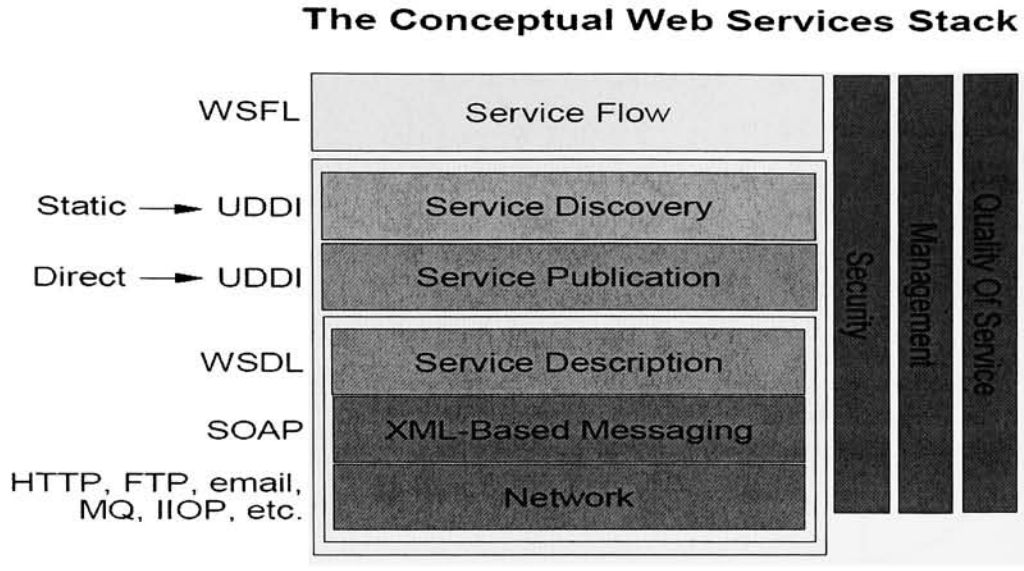

Figure 3-1: Web Services Stack (Kreger 10)

SOAP

Once the input is formatted in XML, the next issue is to send it to the intended Web Service. The simplest way of sending XML data to a Web Service would be to send the XML file by HTTP POST method. But such a simple method is obviously useless as we have to keep the method extensible in order to keep it inter-operable. The current protocol accepted by the big companies like IBM, Microsoft, Sun, etc, is the SOAP (Simple Object Access Protocol). SOAP is basically an HTTP POST with an XML envelop as payload (Kreger 11), but with additional header information. SOAP is preferred over simple HTTP POST of XML, because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of operation or function (Kreger 11).

The SOAP specification was submitted to the W3C by a group of companies including, HP, IBM, Microsoft among others. The definition of SOAP as given in the specification:

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML-based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data-types, and a convention for representing remote procedure calls and responses.

SOAP can be used in combination with or re-enveloped by a variety of network protocols, such as HTTP, SMTP, FTP, RMI over IIOP or MQ (Kreger 13). According to Heather Kreger, most Web Service developers will not have to deal with this

infrastructure directly, but will use optimized programming language-specific bindings generated from WSDL.

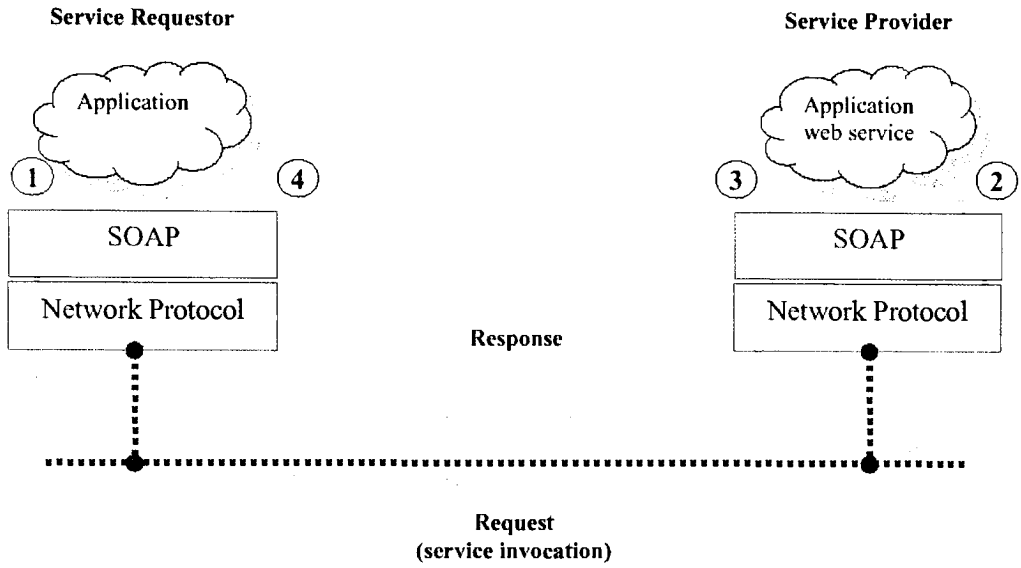

Figure 3-2: High Level Illustration of Web Service communication (Kreger 14)

SOAP defines a way to move XML messages from one point to another. It does this by providing an XML-based messaging framework that is: 1) extensible, 2) usable over a variety of underlying networking protocols, and 3) independent of programming models. It is extensible in the sense that SOAP defines a communication framework that allows for features to be added down the road as layered extensions. SOAP can be used over any transport protocol, such as TCP, HTTP, SMTP, or even MSMQ. SOAP allows for any programming model and is not tied to RPC. SOAP defines a model for processing individual, one-way messages only (Skonnard, par. 4-8). The response to a SOAP message is a separate SOAP message unlike RPC.

SOAP defines five main things: message framing, standardized Fault processing, a way to invoke RPC calls, a mapping to HTTP, and a data-encoding scheme (developMentor 12).

Message Framing:

The SOAP messaging framework defines a suite of XML elements for "packaging" arbitrary XML messages for transport between systems (Skonnard, par 15). The framework consists of four core elements: Envelope, Body, Header and Fault. All these elements are from the "http://schemas.xmlsoap.org/soap/envelope/" namespace.

A typical example of a SOAP message is given in figure 3-3,

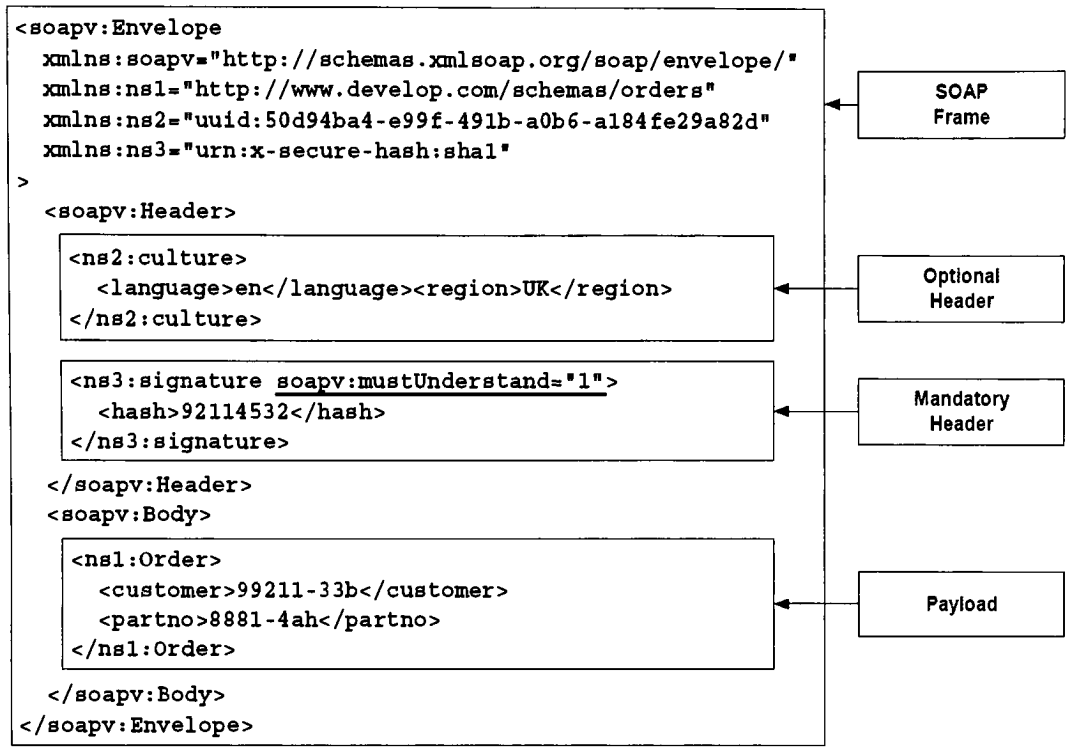

Figure 3-3: SOAP protocol example (developMentor 13)

The Envelope element is always the root element of a SOAP message. The envelop element has one mandatory element called Body and an optional element called Header. Each Header element represents a particular protocol extension and contains contextual metadata that is used to influence the message exchange and/or the exact details of how the message is to be processed (developMentor 13). As shown in the figure 3-3, a header may be optional or mandatory. If a receiver cannot recognize the mandatory header, then it must reject the entire message, as the semantics of the overall message may rely on the correct processing of the mandatory headers (developMentor 13). A SOAP message MUST contain a Body element (W3C SOAP specification). The Body element represents the message payload. The Body element is a generic container in that it can contain any number of elements from any namespace. This is ultimately where the data is kept which is being sent (Skonnard, par 21).

The Fault element is essential, because when some error message is to be sent back by the receiver, the Fault element contains the semantics describing the error message. This makes possible a universal structure of an error message.

So if the following is a SOAP request sent to a financial institution,

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <x:TransferFunds xmlns:x="urn:examples-org:banking">
   <from>22-342439</from>
   <to>98-283843</to>
   <amount>100.00</amount>
```

```
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

The bold part of the message is the payload. And the following is the response sent back.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <x:TransferFundsResponse
    xmlns:x="urn:examples -org:banking">
      <balances>
        <account>
          <id>22-342439</id>
          <balance>33.45</balance>
        </account>
        <account>
          <id>98-283843</id>
          <balance>932.73</balance>
        </account>
      </balances>
    </x:TransferFundsResponse>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Insufficient funds</faultstring>
      <detail>
        <x:TransferError xmlns:x="urn:examples-org:banking">
        <sourceAccount>22-342439</sourceAccount>
        <transferAmount>100.00</transferAmount>
        <currentBalance>33.45</currentBalance>
        </x:TransferError>
      </detail>
      </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

The bold part of the message is the Fault element. The Fault element must contain a faultcode followed by a Faultstring element. The Faultcode element classifies the error using a namespace-qualified name, while the Faultstring element provides a human readable explanation of the error (Skonnard 34). The 'detail' element is optional.

The examples shown were taken from the MSDN article, "Understanding SOAP" by Skonnard, and slightly changed.

SOAP provides a binding to the HTTP protocol. As arguably the most interoperable protocol currently in use, HTTP is a very good fit with the aims of SOAP. Its request-response nature is also well-geared towards using SOAP messages for synchronous RPC-style exchanges. The binding specifies exactly how SOAP messages will be transported over HTTP, including, for instance, which HTTP headers are used, which

HTTP status codes are used and which HTTP verbs are used (developMentor 15). But the SOAP protocol is in no way limited to HTTP protocol. It can support other internet protocols like SMTP, FTP and for intranet purposes, MQSeries, CORBA, etc (Kreger 10-11).

SOAP can be used for transmitting RPC, but the difference is that in SOAP the method call is decomposed into two separate messages: a request and a response. Additionally, SOAP defines a convention for representing the method name, the method parameters and the method return (developMentor, 13).

SOAP makes extensive use of namespacing and attribute specification tags in almost every element of a message. For example when mixing data types within an array you have to set the SOAP-ENC:arrayType to indicate mixed data types within the array in addition to specifying the type of each element of the array (Rhodes, par .8)

Web Service Description, Publication and Discovery:

WSDL (Web Services Description Language):

The technical details of a Web Service are provided in an XML document known as a WSDL document. WSDL stands for Web Services Description Language. WSDL is a machine processable document. WSDL document describes the operations that can be performed by a service. Generally it will also specify the access point for the service, though it is not necessary for a valid WSDL document. The following is a typical WSDL document:

```
  <?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:s="http://www.w3.org/2001/XMLSchema"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:tns="http://tempuri.org/"
      xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
      targetNamespace="http://tempuri.org/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

    <wsdl:types>
      <s:schema elementFormDefault="qualified"
            targetNamespace="http://tempuri.org/">
        <s:element name="GetLog">
          <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="x"
                    type="s:double" />
              </s:sequence>
            </s:complexType>
          </s:element>
        <s:element name="GetLogResponse">
          <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1"
                    name="GetLogResult" type="s:double" />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:schema>
      </wsdl:types>
```

```
<wsdl:message name="GetLogSoapIn">
    <wsdl:part name="parameters" element="tns:GetLog" />
  </wsdl:message>
<wsdl:message name="GetLogSoapOut">
    <wsdl:part name="parameters" element="tns:GetLogResponse" />
  </wsdl:message>

<wsdl:portType name="Service1Soap">
   <wsdl:operation name="GetLog">
       <wsdl:input message="tns:GetLogSoapIn" />
       <wsdl:output message="tns:GetLogSoapOut" />
     </wsdl:operation>
  </wsdl:portType>

<wsdl:binding name="Service1Soap" type="tns:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
   <wsdl:operation name="GetLog">
       <soap:operation soapAction="http://tempuri.org/GetLog"
           style="document" />
     <wsdl:input>
         <soap:body use="literal" />
       </wsdl:input>
     <wsdl:output>
         <soap:body use="literal" />
       </wsdl:output>
     </wsdl:operation>
  </wsdl:binding>

<wsdl:service name="Service1">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
   <wsdl:port name="Service1Soap" binding="tns:Service1Soap">
       <soap:address location="http://imert117/ws/service1.asmx" />
     </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

This WSDL document was created using Visual Studio .NET.

A WSDL document gives information about the operations exposed by the service, the input and output parameters, return values, and the data types used.

A WSDL document has the following major elements (w3schools):

<types>
.... The types used in the service are defined here. For interoperability, types are defined using standard XML schema. The example above is using double data type.
</types>

<message>
.... The example service above supports one operation 'GetLog'. There are two messages corresponding to this operation: 'GetLogSoapIn' and 'GetLogSoapOut'. A message can have many parts, corresponding to the parameters or the return values.
</message>

<portType>
..... The portType element lists all the operations supported by the service and messages within each operation. There can be multiple portType elements corresponding to different methods of invoking the service, e.g. a portType element for SOAP messages, another for HTTP-GET, another for HTTP-POST, etc.

</portType>

<binding>
....Binding elements describe how the messages are to be transported over the wire. There can be multiple binding elements corresponding to SOAP, HTTP-GET, HTTP-POST transports.
</binding>

This constitutes the Service Interface definition. It defines the reusable portions of the service which can be created by multiple independent service implementers. This is analogous to an interface definition in a programming language which can have multiple concrete implementations (Kreger 2001).

A WSDL document can have two more components known as 'service' and 'port'. This portion is the service implementation definition and describes how a service has been implemented. The service element has a collection of port elements where each port element specifies the access point for the Web Service. This is generally a URL or a network address.

UDDI (Universal Description, Discovery and Integration):

UDDI holds the complete Web Service description that includes business name, classification, associated companies, product taxonomy, quality of service, physical location, etc along with the WSDL. A UDDI entry has basic data structures as shown in figure 3-4:

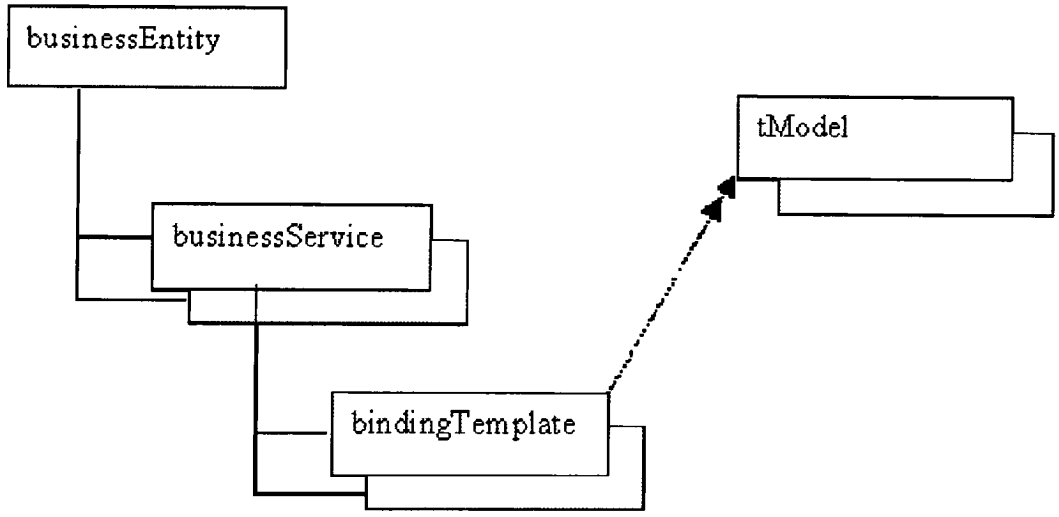

Figure 3-4: UDDI Business Entity

businessEntity element contains information about the business, like name, address, contact information, etc., the classification of the company and identification information. This element cam contains multiple businessService elements. Each businessService element contains information about a Web Service being published by this company. It may in-turn contain multiple bindingTemplate elements which

specifies the access point and describes the technical specification that this Web Service is following. A technical specification is modeled as a 'tmodel'.

Publication and Discovery of Web Service:

A Web Service can be published in the most low-tech form of an email attachment or CDROM distribution of the WSDL file, i.e. direct publish from the service provider to the service requestor. Or a service provider can publish the Web Service description to a UDDI node at the internal enterprise level or a more public UDDI node like uddi.microsoft.com.

Web Service requestors can retrieve Web Service description from a Web Service repository or a UDDI node. The criteria for deciding a proper Web Service could be prior agreement with a particular provider, type of interface, quality of service, physical distance, binding protocols, service classification, business information, etc. When a service requestor has to dynamically select a service provider, a policy for choosing one from many providers will have to be determined if a public UDDI node is used. In case of an internal UDDI node, all services are trusted and service selection can be based on binding support, quality of service information, etc.

Once a service has been selected and the service description acquired by the requestor, now the requestor needs to process it to invoke the service. The requestor needs to create SOAP request or proxy classes specific to the programming language. This can be done at design time, or even at runtime. Various tools exist to create these proxies (e.g. wsdl.exe for .NET) which can be used at runtime, to create programming language bindings to the WSDL document which hide the XML messaging details from the programmer.

The following code in VB gives an example of how to find a list of Web Services from a UDDI which follow a particular tModel. This way the application can find a list of compatible Web Service providers and can invoke any one of these Web Services depending on other criteria, like trustworthiness, business rating, quality of service, etc.

```
Private Sub Find_Services(string tModelKey)

        ' do a findService with the tModelKey
        ' this will return a list of services that implement
        ' the tModel -- we will then have to iterate over
        ' that list and make additional calls to UDDI

        Dim fs As New FindService()

        ' pass the interface tModel key
        fs.TModelKeys.Add(tModelKey)

        ' pass an empty BusinessKey so that we search across all
providers
        ' in the UDDI Services registry
        fs.BusinessKey = ""
        Dim sl As New ServiceList()

        Try
            sl = fs.Send()
```

```
        Catch ue As UddiException
            MessageBox.Show(ue.StackTrace)
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try

        ' now that we have the list of services that do in fact
implement
        ' that tModel, we can issue a find_binding,
        ' passing the appropriate serviceKey

        Dim fbind As New FindBinding()
        Dim bindd As New BindingDetail()
        fbind.TModelKeys.Add(tModelKey)

        Dim si As ServiceInfo
        For Each si In sl.ServiceInfos

            fbind.ServiceKey = si.ServiceKey
            Try
                bindd = fbind.Send()
            Catch ue As UddiException
                MessageBox.Show(ue.Message)
            Catch ex As Exception
            MessageBox.Show(ex.Message)
            End Try

            'grab the accessPoints
            Dim bt As BindingTemplate
            For Each bt In bindd.BindingTemplates

               //here bt.AccessPoint will give you the access point
               //of this Web Service. This can be further used as
               //required by the application.

            Next bt
        Next si
    End Sub
```

This code was modified from the code given in the article "Using UDDI at Run Time, Part II" (Januszewski 2001).

UDDI is one step towards achieving an E-marketplace where contracts can be made between a service provider and service requestor dynamically and reliably. Service providers can be changed at the last minute without any human intervention. This is allowed by the ubiquity and nonproprietary nature of technologies used, standardized operating procedures and all big players taking an active part in creating Web Service specifications. Independent groups, like Open Application Group, create standards which can be adopted industry-wide which will create more interoperability among businesses operating in the same industry.

References

Ananthamurty, L. (2002 October 22). Introduction to Web Services. *Developer.com* Retrieved January 11, 2005 from http://www.developer.com/services/article.php/1485821

Connell. "Web Services in Action: Aligning IT with Business Objectives". WebService.org. (2003 October) < http://www.webservices.org/index.php/article/articleview/1152/1/7/>

DevelopMentor Web Services Team, "Understanding Web Services", (2003 October) < http://www.develop.com/downloads/WebServicesLGFLN.pdf>

DevelopMentor Web Services Team. (2002). Understanding Web Services. Retrieved October 18, 2003 from http://www.develop.com/downloads/WebServicesLGFLN.pdf

Gralla, Preston. "An inside look at Web services and enterprise applications", (2003 September) <http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci1005337,00.html>

IBM Glossary. "Web Services by IBM: Glossary" (2003 October) <http://www-3.ibm.com/software/solutions/webservices/glossary.html>

Januszewski, K. (2001, December) Using UDDI at Run Time, Part II. Microsoft Corporation. Retrieved from http://msdn.microsoft.com/library/en-us/dnuddi/html/runtimeuddi2.asp

Kreger. "Web Services Conceptual Architecture". IBM.com. (2001 May). <http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>

Rhodes, Kate, "XML-RPC vs. SOAP", (2003 October) <http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm>

Schmelzer, R quoted by Trotta, G. (2004 May 1) SOA and Web Services: Bridging Barriers To Agile Integration. *ebizQ* Retrieved January 11, 2005 from http://www.ebizq.net/topics/soa/features/3521.html

Shah, Rawn "Start here to Learn about Web Services". IBM.com (2003 October) <http://www-106.ibm.com/developerworks/library/ws-starthere.html>

Skonnard, A (2003 March). Understanding SOAP *DevelopMentor* Retrieved from http://msdn.microsoft.com/library/en-us//dnsoap/html/understandsoap.asp

Skonnard, Aaron "Understanding SOAP" DevelopMentor (2003 March) <http://msdn.microsoft.com/library/en-us//dnsoap/html/understandsoap.asp>

SOAP v1.2. (2003). SOAP v1.2. Retrieved from http://www.w3.org/TR/soap12-part1/

Uanny, IDL-XML Based Information Sharing Model for Enterprise Integration July 2000 Diss. Rochester Institute of Technology, 2000

W3C SOAP Specification, "Simple Object Access Protocol (SOAP) 1.1". (2000 May) < http://www.w3.org/TR/SOAP/>

W3C. "Web Services". World Wide Web Consortium. (2003 October) < http://www.w3.org/2002/ws/>

W3Schools (2005). WSDL Tutorial. Retrieved February 1, 2005 from http://www.w3schools.com/wsdl/

4. Web Services versus Earlier Technologies:

There have been numerous technologies so far that have allowed communication between two software programs one way or the other, like RPC, ORBA, RMI, etc. But the concept of Web Services has gained a lot more popularity than these earlier technologies, because of certain important technical and some fundamental philosophical differences in design. The following tries to highlight these differences.

RPC

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details (searchWebServices).

The client program has a stub included in the compiled code, which receives the remote procedure call from the client program and forwards it to a runtime program on the client machine. This runtime program has the knowledge of network details and passes on the procedure call to the remote machine. The runtime on the remote machine receives this request and passes on the request to the server program via the stub that is included in the compiled code of the server program. The results are returned the same way.

(Vondrak, 1997)

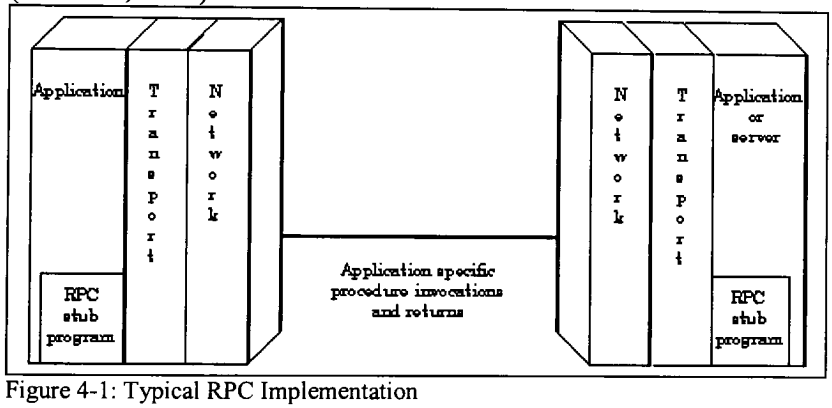

Figure 4-1: Typical RPC Implementation

By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are located on the same system. However, RPC increases the involvement of an application developer with the complexity of the master/slave nature of the client/server mechanism (Vondrak 1997). RPC is the oldest attempt of distributed programming. Most RPC implementations do not support asynchronous calls or peer to peers calls. This makes RPC unsuitable for object-oriented programming. For synchronous calls to work, the client and the server have to be always available and functioning (i.e. the server should not be blocked in some other operation). Otherwise to recover from a blocked condition, the implementation needs to provide mechanisms, such as error message, retransmission, redirection to an alternate server, etc. RPCs that implement asynchronous mechanisms are very few and complex to implement (Vondrak 1997)

The synchronous nature of RPC helps in maintaining the load on the network at low levels. However if RPC implements recovery mechanisms, like retransmissions, the load on the network will increase. Also, RPC uses static routing tables established at compile time; hence performing load balancing across network becomes difficult and should be considered while designing an RPC application (Vondrak 1997)

RPC works in a way very similar to Web Services. In fact the term XML-RPC is loosely associated with Web Services (Ethan Cerami, 2002). There have been attempts at standardization of RPC protocol, but more than one standardized systems have been created (Wikipedia, 2004). Distributed Computing Environment (DCE) is one popular model (searchWebServices). There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols (Wikipedia, 2004).

In RPC, the server and client systems need to know at design time what the data structure is going to be like; this results in tightly coupled clients and servers (Ron Zahavi, 1999). Proprietary implementations of RPC protocol add to this problem. Because of the complexity of the synchronous mechanism of RPC and the proprietary and unique nature of RPC implementations, training is essential even for the experienced programmer (Vondrak 1997)

EDI

EDI (Electronic Data Interchange) is an electronic means for companies to exchange business documents (Jilovec, 1998). By the very purpose of this technology, trading companies have to agree on what information will be sent. Larger companies will force their rules about the format in which specific information is to be transmitted, on their smaller partners. There can be various standards used for representing the data electronically, but three are most popular: UN/EDIFACT (United Nations/Electronic Data Interchange For Administration, Commerce and Transport) (outside US) and ANSI (American National Standards Institute) ASC X12 and Uniform Communications Standard (UCS) in US. The term "EDI" is used for the global international standard and the term "edi" is used for non-standard and proprietary implementations. (Wikipedia, 2004).

EDI is expensive for smaller companies, because to do business with many other companies they have to do different mapping of fields of information in their electronic documents for each of the partner. Using Web Services can solve this problem, because an XML schema can have a standard definition and, at the same time can have controlled extensions, which can be used to describe specific business context (Burdett, 2002).

CORBA

Common Object Request Broker Architecture (CORBA) is an architecture and specification for creating, distributing, and managing distributed program objects in a network (searchDataBase). Similar to RPC, CORBA involves an intermediary

program known as ORB (Object Request Broker), which conveys the client request to its appropriate destination so that the client program does not have to understand the network details. ORB is more sophisticated than the similar concept used in RPC. CORBA uses an interface definition language (IDL) to specify the interfaces an object will expose to the world (Wikipedia, 2005).

CORBA automates many common network programming tasks, such as: object registration, location, and activation; request demultiplexing; framing and errorhandling; parameter marshalling and demarshalling; and operation dispatching (Schmidt 2004). The figure 4-2 illustrates the primary components in the CORBA ORB architecture:

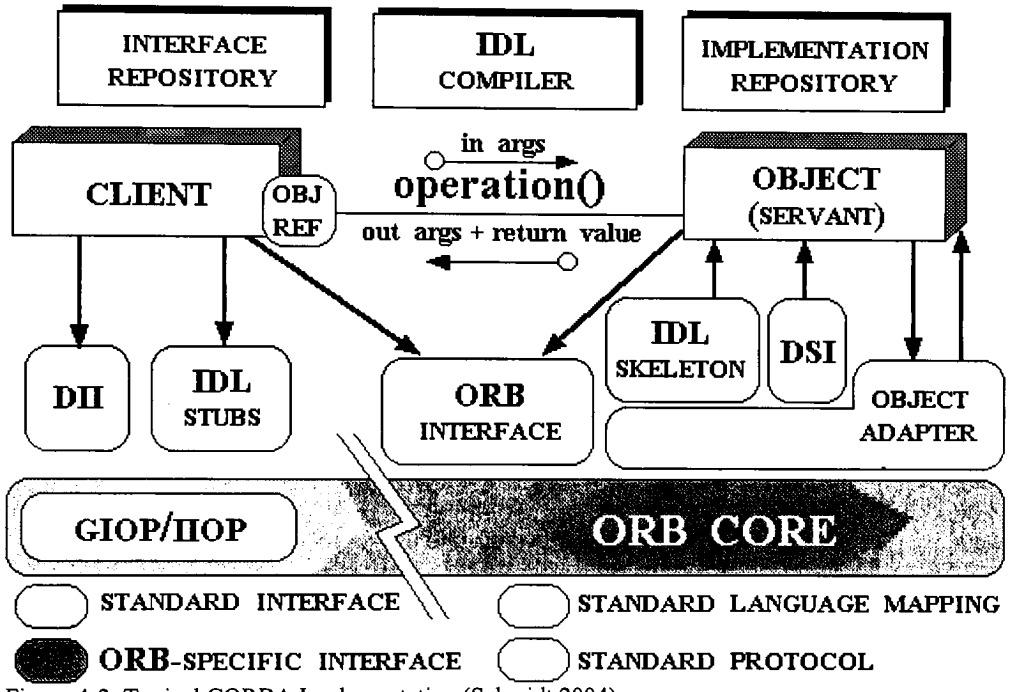

Figure 4-2: Typical CORBA Implementation (Schmidt 2004)

The following is from (Schmidt 2004) and (Keahy 1998):

Object – This is a CORBA concept, which consists of an identity, an interface and implementation. The implementation is also known as 'servant.'

Servant – This is the actual implementation of the remote object, which can be written in any language, like C, C++, Java, SmallTalk, etc. This entity defines operations that support the CORBA IDL.

Object Request Broker – As mentioned above, ORB defines mechanism for transparently communicating between the client and the server. The calls to the remote object appear like local calls, because of the use of ORB. ORB simplifies distributed programming by taking care of the actual network communication itself rather than having the client do it.

ORB Interface – Since ORB can be implemented in any number of ways, to have a standard way of communicating to the ORB, CORBA specification defines an abstract interface for an ORB.

IDL Stubs and Skeleton – IDL stub represents the mapping between the language of implementation of the client and the ORB, similarly IDL skeleton link the server implementation and the ORB. The client and the server can be written in any language as long as the ORB implementation supports this mapping between the implementation language and the ORB. The transformation between CORBA IDL definitions and the implementation language is done by a CORBA IDL compiler.

Dynamic Invocation Interface (DII) – Clients can use DII to issue requests to the ORB by bypassing the IDL stub. Unlike synchronous (blocking) calls through the IDL stubs, the DII allows for non-synchronous calls and one-way calls. DII also allows clients to specify requests to objects whose definition and interface are unknown to the client at the compile time.

Dynamic Skeleton Interface – This is a server side analogue to DII. DSI allows an ORB to deliver requests to an object implementation, which does not have compile time knowledge of the client that it is implementing. The client making the request does not need to know whether the ORB uses type-specific IDL skeletons or the dynamic skeletons.

Object Adapter – Object Adapter assists the ORB with delivering requests to the object. Object Adapter is responsible for generating and interpreting object references, method invocations, security of interactions, object and implementation activation and deactivation, mapping references corresponding to object implementations and registration of implementations. OA can be specialized to provide support for certain object implementation styles.

Following is a simple example of using CORBA from (Keahy 1998).
Say, a remote object is capable of performing an operation: 'operation_on_object,' which takes a long integer and a boolean value as input. The boolean value is an out parameter as well. The programmer needs to write an IDL specification of the object and the operations that it is supposed to perform, for example:

```
interface example_object
{
      void operation_on_object(in long a, inout boolean b);
}
```

This IDL specification is then passed through an IDL compiler, which will generate stubs and skeletons for client and server, respectively, in the language that the client and the server are written.

The following is the 'binding' code on the client side.

```
class example_object: public virtual CORBA::Object
{
static example_object_Ref _bind(...);
```

```
virtual void operation_on_object(CORBA::Long a,
CORBA::Boolean& b);
}
```

The function 'operation_on_object' in this class represents a stub. Its implementation will take care of sending the parameters 'a' and 'b' to the server and wait for an answer. The '_bind' function will bind to a specific object of the 'example_object' type, given some of its characteristics. This code is automatically generated by the compiler.

The following code goes on the server side.

```
class example_object_sk: public virtual example_object
{
virtual void operation_on_object(CORBA::Long a,
CORBA::Boolean& b) = 0;
static void call_operation_on_object( /* a stream */);
};
```

The upcall from the Object Adapter will be made through the 'call_operation_on_object' function. This function will unmarshal the parameters and then execute the function operation_on_object, which implements the service. Since this is a pure virtual function, it has to be implemented in a derived class as shown below:

```
class example_object_impl: public example_object_sk
{
void operation_on_object(CORBA::Long a, CORBA::Boolean& b);
};
```

This class contains the actual implementation of the functions, and it has to be written by the programmer.

In the simplest scenario, main program of server implementation contains an instantiation of the implemented object and a call to the basic Object Adapter, telling it that the implementation is ready:

```
CORBA::BOA::impl_is_ready();
```

The client can now use this remote function as shown below:

```
example_object* x = example_object::_bind("my_example_object",
<host_name>);
x->operation_on_object(4,0);
```

As seen in this code above, CORBA necessitates certain keywords to be used in the programming, which are specific to CORBA. In the case of Web Services coding, there is no such requirement. Also, the implementation of ORB imposes certain restriction on which languages can be used to implement the client and the server. During the initial years of CORBA till CORBA 2.0, the different ORBs were incompatible with each other. CORBA also results in tight coupling, because the client of a particular service needs to use the same classes as server for method

parameters (Shah, 2004). Adopting CORBA necessitates major changes to existing applications and infrastructure (Greenfield, 2003). This technology is most suitable to work over the intranet (Shah, 2004).

RMI

Java Remote Method Invocation (RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism (Sun Microsystems, Inc. 2004).

Just as in CORBA, RMI applications involve a client and a server with the server creating some remote objects, making their references accessible and waiting for the clients to invoke methods on them. RMI provides the mechanism by which the server and the client communicate.

(Sun Microsystems, Inc. RMI Tutorial, 2004) An application can register its remote objects with RMI's naming facility known as 'rmiregistry.' Client applications can then obtain the reference by communicating with the rmiregistry. The applications can also pass and return object references as part of their normal operations. Details of network communication between client and server are handled by RMI. To the programmer, calls on remote objects will appear like normal java invocations. RMI has a unique and powerful feature of dynamic code loading: A virtual machine can use a class entirely unknown to it, by downloading the code of that class.

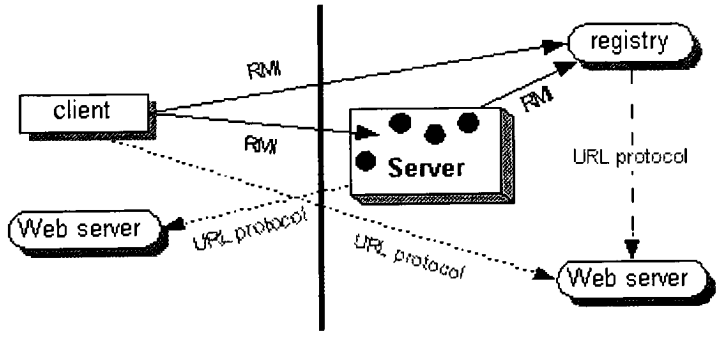

Figure 4-3: RMI Communication (Sun Microsystems, Inc RMI Tutorial 2004)

This image depicts how a client can obtain reference of a remote object by going to the registry and also shows how a client and server can load code of unknown object from each other's web servers and use those classes.

The advantage of Dynamic Code Loading is that the type and behavior of an object previously known to only a single virtual machine can now be transmitted to another virtual machine. RMI passes the objects by their true types; hence, their behavior is not changed when they are sent to another machine.

The following is an example of how the RMI Code looks (Reilly, 2000)

Say, our server provides the capability of squaring a given number. We have to write an interface that extends the 'remote' interface as shown below:

```
import java.math.BigInteger;
import java.rmi.*;

//
// Interface for a RMI service that calculates powers
//
public interface PowerService extends java.rmi.Remote
{
public BigInteger square ( int number ) throws
RemoteException;
}
```

This interface is implemented by a class called PowerServiceServer which is our server:

```
import java.math.*;
import java.rmi.*;
import java.rmi.server.*;

//
// PowerServiceServer
//
// Server for a RMI service that calculates powers
//
public class PowerServiceServer extends UnicastRemoteObject
implements PowerService
{
    public PowerServiceServer () throws RemoteException
    {
        super();
    }

    // Calculate the square of a number
    public BigInteger square ( int number )
    throws RemoteException
    {
        String numrep = String.valueOf(number);
        BigInteger bi = new BigInteger (numrep);

        // Square the number
        bi.multiply(bi);

        return (bi);
    }


    public static void main ( String args[] ) throws Exception
    {
        // Assign a security manager, in the event that
dynamic
// classes are loaded
```

```
        if (System.getSecurityManager() == null)
            System.setSecurityManager ( new
RMISecurityManager() );

        // Create an instance of our power service server ...
        PowerServiceServer svr = new PowerServiceServer();

        // ... and bind it with the RMI Registry
        Naming.bind ("PowerService", svr);

        System.out.println ("Service bound....");
    }
}
```

As can be seen, the main method of the server creates an instance of the server and binds it to the RMI registry. It will also assign a security manager to the virtual machine.

The following is the code for client:

```
import java.rmi.*;
import java.rmi.Naming;
import java.io.*;

//
//
// PowerServiceClient
//
//
public class PowerServiceClient
{
public static void main(String args[]) throws Exception
{

String hostName=args[0];

// Assign security manager
if (System.getSecurityManager() == null)
{
System.setSecurityManager
(new RMISecurityManager());
}

// Call registry for PowerService
PowerService service = (PowerService) Naming.lookup
("rmi://" + hostName + "/PowerService");

  System.out.println
  ("Square of 75 : " + service.square(75));

}
```

In the client code, the client will first assign a security manager if one is not already present. An instance of the service is created by calling the registry and locating the

service. The client receives an instance of the interface that was defined earlier and not the actual implementation.

The key differences between RMI and CORBA are as follows (Baclawski, 1998):

1. CORBA is a language-independent standard, whereas RMI is a feature of java only.
2. CORBA includes many other capabilities which are not found in RMI.
3. An "Object Request Broker" is not required in RMI.

Java RMI is trying more and more to be compatible with CORBA. There is a new form of RMI called RMI/IIOP which uses CORBA's IIOP as the underlying protocol for RMI communications.

Web Services

Web Services are software programs, which have the capacity to interact with any other software programs regardless of the language in which the two programs are written as long as the two software programs adopt the same standards of communication. The idea of Web Services is to allow communication between two computer-programs without any human intervention. Web Services are self contained, self-describing, modular applications that can be published, located and invoked across the web (IBM Glossary).

The motivations behind the Web Services are more from the Business point of view than from pure technical point of view. Web Services enable loose coupling between a service provider and a service requestor, So in an ideal Web Services world, a service requestor can change its service provider without much of investment if it does not like the service provider's quality of service. In case of, say, CORBA or RPC, doing something like this is very expensive, because it is necessary for the service requestor to invest in an entire software infrastructure that would support the software used by the service provider. This is called tight coupling between applications. But Web Services hide the inner software infrastructure from the outside world and use the standard XML over SOAP for communication. Hence there is no compulsion to use any specific software. Also the modification in one of the party's software infrastructure does not have any impact on the communication.

Also one big difference between technologies like CORBA, RPC, RMI and Web Services is that, while in these earlier technologies focus is more on the distributed object programming, in Web Services the focus is completely on the business aspect and not on the software aspect at all. In the recent months, Web Services have boosted a growth in interest in 'Service-Oriented Architecture' (Gralla, 2004).

Web Services can be used as an extension to legacy enterprise applications making them more interactive with external applications.

Web Services Infrastructure (MSDN 2005):

XML Web Services Directories provide a central location for searching Web Services offered by other organizations. UDDI registries fulfill this role. Clients to a service may or may not need to use the directory.

XML Web Service Discovery is the process of discovering one or more related documents that describe a Web Service. The DISCO specification specifies an algorithm for achieving this. If clients already know the location of the service then this step is unnecessary.

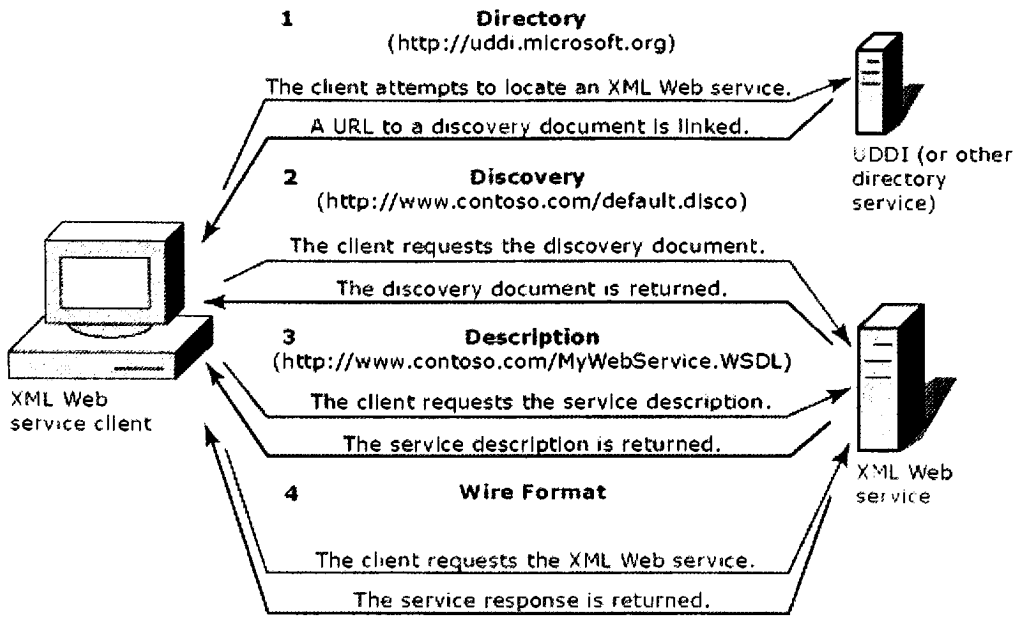

Figure 4-4: XML Web Services Infrastructure (MSDN 2005)

Web Service Description defines the interactions the Web Service supports. This is necessary to know in order for a client to interact with the service. This description is provided in a machine-processable format known as WSDL – Web Service Description Language.

Wire Formats: XML Web Services use open wire formats, i.e. the most common open web standards in order to enable universal communication. SOAP is the standard protocol used for communication.

The only limitation that Web Service specifications put on the way the service and the clients are implemented is that they use SOAP for communicating. SOAP is basically an HTTP POST with an XML envelope as payload (Kreger 2001), but with additional header information. SOAP is preferred over simple HTTP POST of XML, because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of operation or function (Kreger 2001). The network protocol over which the SOAP messages are carried can be HTTP, SMTP, FTP, RMI/IIOP or anything else, although HTTP is the recommended one, because of its ubiquity and, because it is non-proprietary. Most Web Service developers will not have to deal with this infrastructure directly, but will use optimized programming language-specific bindings generated from WSDL.

Table 4-1 gives a few points distinguishing Web Services from CORBA and Java RMI.:

Table 4-1: Points of distinction between Web Services, CORBA and RMI(Shah, 2004)

	Web services	CORBA/IIOP	J2EE (EJB/RMI)

	Web services	CORBA/IIOP	J2EE (EJB/RMI)
Best Suited for Network type	Internet (characterized by high latency and n/w failures)	Intranets with characteristics of high speed and high Reliability	Intranets
Look up mechanism	Description of service in human readable form	Single unique ID (No description of the service)	Single unique id
Data exchange format	XML (Human and machine readable/ translatable)	Programming language objects (Non-Readable)	Objects
Dynamic Integration	High	Requires understanding of IDL and shipping of proxy	Requires understanding of IDL and shipping of proxy
Level of coupling	Loose. (XML, and is self-explanatory and extensible).	Tight coupling (The client using the service needs to use the same classes as server for method parameters)	Tight coupling
Extensibility	Polymorphism possible by use of optional meta tags in the XML data	Polymorphism across network not possible	

Concluding, the major reasons for popularity of Web Service compared to these earlier technologies are:

- There is no new invention in Web Services. Web Services leverage open standards and protocols. Web Services use the HTTP protocol, and XML as the base language. These are well-known and successful technologies, unlike the other technologies which involve new concepts to learn (Ananthamurty, 2002)
- Web Services are based on standardized rules, which make them more portable. If a system can connect to one Web Service, then it can connect to any other Web Service by any other provider. Web services become portable, because of these standards (Ananthamurty, 2002).
- Web Services are not just standard-based APIs, but instead these are analogous to actual business services (Schmelzer, 2004) that can be provided to create a loosely coupled business architecture. This increases the excitement around Web Services.

Web Services have the backing of almost all major players, like Microsoft, IBM, and Sun. The standards for Web Services are not proprietary.

References

Baclawski, Ken. (1998). Java RMI Tutorial. Retrieved January 18, 2005 from http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

Burdett, David. (2002, December). Avoiding EDI's Mistakes. *eAI Journal*., p 8, Retrieved January 11, 2005 from http://www.eaijournal.com/PDF/Burdett.pdf

Cerami, Ethan. (2002, February). Web Services Essentials. p 4, O'Reilly & Associates, Inc.

Gralla, P. (2004 November). SOAs gaining momentum, Part 1. Retrieved January 18, 2005 from http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1022911,00.html

Greenfield, P. (2003 March) Web Services – Facts, Fables and Future. Retrieved January 11, 2005 from http://ict.csiro.au/MU/Trends/Presentations/CSIROWebServices.pdf

IBM Glossary. (n.d.) Web Services by IBM:Glossary. Retrieved January 18, 2005 from http://www-3.ibm.com/software/solutions/webservices/glossary.html

Jilovec, Nahid (1998). The A to Z of EDI. p xv, Duke Press

Keahy, Kate. (1998). A Brief Tutorial on CORBA. Retrieved January 18, 2005 from http://www.cs.indiana.edu/~kksiazek/tuto.html

Kreger. (2001 May). Web Services Conceptual Architecture. Retrieved from http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

MSDN (2005). XML Web Services Infrastructure. http://msdn2.microsoft.com/en-us/library/sd5s0c6d.aspx

Open Applications Group. http://openapplications.org/index.htm

Reilly, David. (2000, April). Introduction to Java RMI. Retrieved January 18, 2005 from http://www.javacoffeebreak.com/articles/javarmi/javarmi.html

Schmidt, D. (2004 May). Overview of CORBA. Retrieved January 18, 2005 from http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

searchDatabase.com definitions (n.d.). Retrieved January 11, 2005 from, http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci213865,00.html

searchWebServices.com definitions (n.d.). Retrieved January 8, 2005 from, http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci214272,00.html

Shah, G.B. (2004 November, 29) *DeveloperIQ*. Retrieved January 11, 2005 from http://www.developeriq.com/articles/view_article.php?id=378

Sun Microsystems, Inc (2004). Java Remote Method Invocation. Retrieved January 18, 2005 from http://java.sun.com/products/jdk/rmi/

Sun Microsystems, Inc RMI Tutorial (2004). An Overview of RMI Applications. Retrieved January 18, 2005 from http://java.sun.com/docs/books/tutorial/rmi/overview.html

Vondrak, Cory (1997 June 25). Remote Procedure Call. Retrieved January 18, 2005 from http://www.sei.cmu.edu/str/descriptions/rpc.html

Wikipedia, the free encyclopedia. (2004, December 25). Remote Procedure Call. Retrieved January 11, 2005 from http://en.wikipedia.org/wiki/RPC

Wikipedia, the free encyclopedia. (2004, December 29). Electronic Data Interchange. Retrieved January 11, 2005 from http://en.wikipedia.org/wiki/EDI

Wikipedia, the free encyclopedia. (2005, January 10). CORBA. Retrieved January 11, 2005 from http://en.wikipedia.org/wiki/CORBA

Zahavi, Ron. (1999). Enterprise Application Integration with CORBA. p xxxix, Wiley Computer Publishing.

5. Web Services Case Studies:

Before getting into the technical details of Web Services, we will take a look at how Web Services are used in real life. By looking at the Web Service implementations carried out by Microsoft within various organizations, we will see that most organizations employ Web Services for sharing their services with their partners, sharing their databases with their partners, or for wrapping their existing web applications with XML and SOAP so as to give them more flexibility and extensibility.

As an introduction, consider an imaginary bookstore, XYZ, which has been selling books traditionally and wants to create a website from which customers can order books. Now, instead of creating an entire IT infrastructure for making a database of books and a system of keeping track of orders, this bookstore can use the Web Service provided by, say, *Barnes and Noble*. Customers can visit the website of the XYZ store and order books, but the orders will be managed by Barnes and Noble. In this way, the bookstore still has the advantage of its own brand image and flexibility in terms of the layout of their website, but they do not incur any of the headaches of managing customer orders. Barnes and Noble is sharing its services and database with the XYZ store, but now that the XYZ store is using a Web Service, it isn't really tied to Barnes and Noble. If it happens that XYZ is not happy with the service provided by Barnes and Noble and wants to switch over to, say, *Amazon*, it can do so by making only minimal changes in its system. All of this will happen in a manner that is completely transparent to the customers of XYZ's website, provided that Amazon offers all of the same features as *Barnes and Noble*.

Some real life examples are explained below:

Common Picture eXchange environment (CPXe)

CPXe is an industry initiative to implement "An open, 'Common Picture eXchange environment' whereby imaging devices can seamlessly exchange digital images, order and commerce information among any networked imaging applications and services, regardless of manufacturer, service provider or geography" (Giantsopoulos, Slide 8).

The CPXe initiative is an ambitious project spearheaded by the Kodak Company. Focusing on the Web Services aspect of it, CPXe is going to create a network to connect the array of services offered by all types of providers. CPXe is also going to create a master service directory, a Web Service, which makes all of those services easily available to anyone. The member companies will provide services that plug-in, e.g. a retail printing service with applications that use the services on the network (Giantsopoulos, Slide 14).

The services provided by the member companies are stored in a common registry from which searching for these services will be very easy. The registry is known as UDDI (Universal Description, Discovery and Integration) and this is one of the first actual uses UDDI technology (Smith and Heuter, 2003).

The software accompanying every digital camera or downloadable software from the manufacturer's site will present the user with the types of service that he can make use of. This software will communicate with the CPXe central service directory and retrieve a list of retail locations or online services which the user can use for printing the images or some similar type of service.

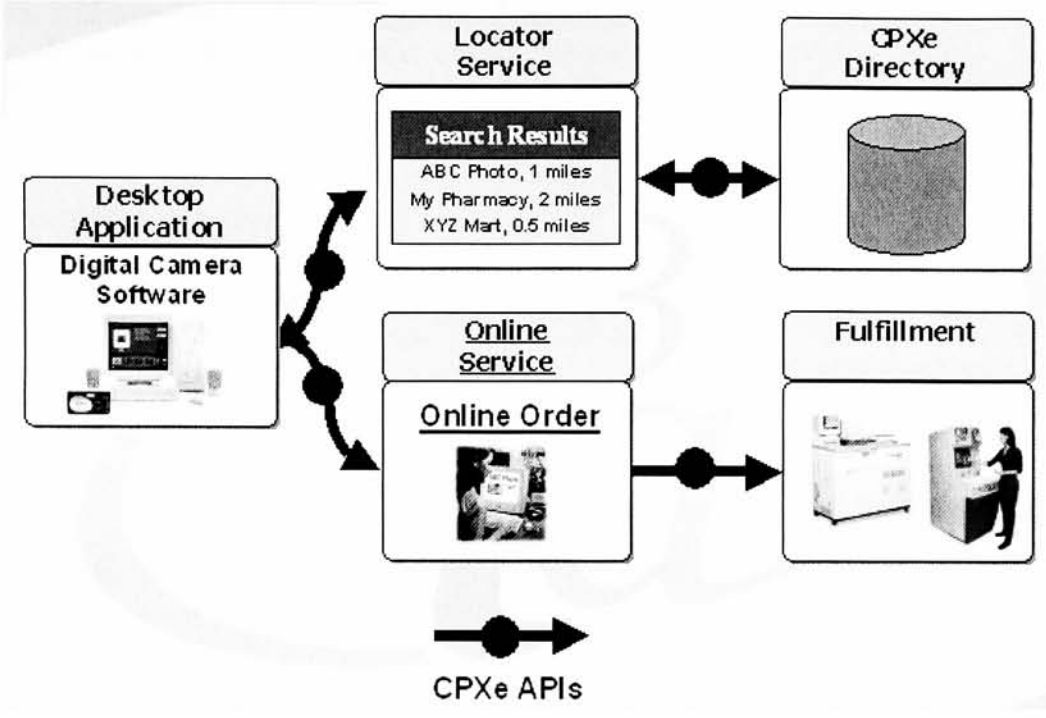

Figure 5-1: Use of the CPXe central directory (Giantsopoulos, Slide 11)

Zagat Survey

(Microsoft)

Zagat Survey provides restaurant guides for 45 regions worldwide, as well as guides to airlines, hotels, resorts, spas, nightlife, and food and entertaining resources. Visitors to the company's website have access to restaurant guides and surveys.

Zagat makes its restaurant reviews available on partner websites in order to increase its visibility on the internet. To achieve this, Zagat used to export its database and make copies of it available to its partners on a monthly basis. This required a full time Database Administrator (DBA) converting the database into various formats as required by the various partners. This creates an additional workload for Zagat and partners need to allocate additional resource for updating their databases every month. Each new partner means an increase in both of these hassles.

Web Services came as a solution to these problems. Using Web Services, Zagat has exposed its database to its partners. Each time a partner site wants to show a Zagat review, it will query the Zagat database through the Web Service developed by Zagat. This provides multiple benefits: The cost of one DBA is saved for Zagat; No more

monthly updates required for partner sites; They will no longer need to build the infrastructure to import, store, and manage the database; The partner sites will always have access to the latest surveys and reviews from the Zagat website; And, in the aspect where the Web Services will help the most, it will provide the Zagat Company with more flexibility in striking new partnerships and growing its revenue. Instead of licensing the database to other companies on a monthly basis, Zagat can now charge the companies on various bases, such as a flat monthly fee or per search fee. Additionally, adding a new partner will not increase the hassles of distributing the database copies and other necessary resources. The new partners will simply obtain the data from the Zagat Web Service.

Dollar Rent A Car

(Microsoft)

Dollar Rent A Car is one of the world's largest car rental agencies, with more than 400 locations in 26 countries. The company uses a VMS (Virtual Memory System) based application known as Quick Keys as its reservation system. In the past, the company received reservation data that had been entered into the Quick Keys application by tour operators. Dollar had three ways if achieving this. The first was an internally developed EDI (Electronic Data Interchange) interface which parsed flat files that were sent to Dollar on a daily basis by several tour operators via FTP (File Transfer Protocol). The second path was through *dollar.com*. And, finally, a majority of its reservations were received through Global Distribution Systems (GDS), such as Sabre or Apollo, which require a fee for each transaction.

If Dollar could provide its partners with direct access to its reservation system, it would increase its incremental reservations and, at the same time, reduce its customer acquisition cost. Within Dollar's Advanced Technology Group, the initial consensus was that it would be better to integrate partners through *dollar.com* rather than attempting to connect directly to the mainframe. Unfortunately, the existing connectivity between *dollar.com* and Quick Keys could not be easily adapted to solve this attractive business opportunity.

After weighing the pros and cons of many available options, Web Services were selected for achieving the objective. The technical team had many concerns; one being the fact that XML uses text-based tags to describe fields—a characteristic Dollar was afraid would enlarge each message and impact its transfer time. Another concern was that a Web-based solution might not be able to handle the proposed transaction volumes. The final issue was the newness of XML Web Services—some people at Dollar were uncomfortable transitioning to a new technology. However, after Microsoft convinced Dollar that the expected transaction volume could be supported and, because Web Services offered inherent advantages like flexibility and extensibility, the Web Service option was selected.

After the Web Service was put into place, the partners could now send the reservation requests to Dollar using SOAP requests. At the Dollar datacenter, a solution residing on the Microsoft platform exposed the pre-existing COM-based interface to Quick Keys as an XML Web service.

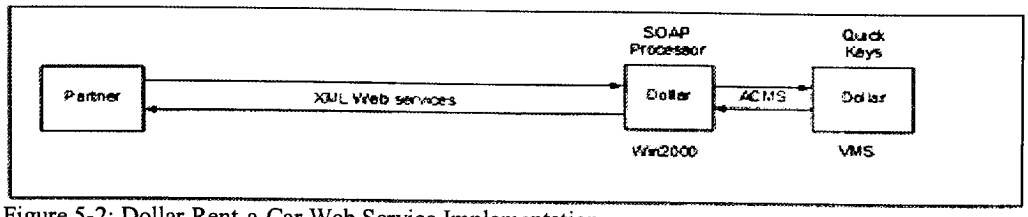

Figure 5-2: Dollar Rent-a-Car Web Service Implementation

By integrating directly with business partners using the XML Web Services, Dollar cost-effectively opened up another sales channel that has provided millions of additional rate requests and thousands of new reservations per year, equating to millions of dollars in additional revenue. Furthermore, the company expects to save a significant amount in annual GDS fees. Just as important, the company put in place a simple, re-usable interface into its reservation system that could be accessed by any system via the Internet. When the company needed to open up reservation access to Palm users, all that was needed to do was to develop a user interface for the Palm, relying on the existing XML Web service interface to provide real-time connectivity with the Quick Keys application running on the mainframe.

When it comes to actually deploying or consuming a Web Service, many enterprises accept the help of System Integrators and Consultants. Gartner end user research shows that Supply Chain Management is the most popular area of Web Services deployment, because Web Services can integrate with the existing legacy applications of an organization as well as with the partners' and suppliers' systems (Cantara, 2003).

It is also observed that, although many enterprises are using XML in Systems Integration projects, few are using SOAP along with it. This is called the XML veneer approach, which is not going to be as beneficial as Web Services in the long run. The veneer approach is common in spite of the fact that Systems Integration vendors are encouraging enterprises to use SOAP in their projects. Also, very few enterprises are using WSDL and UDDI standards (Cantara, 2003). Both WSDL and UDDI are required only when one Web Service needs to interact with another Web Service. Since this might not be the case in intra-enterprise Systems Integration, WSDL and UDDI are infrequently used.

The Gartner research also shows that internal Web Services are giving way to external Web Services (Cantara, 2). This might be, because it is probably not worth the trouble to use Web Services within a single company since custom applications can be much better suited.

Going back to our very first example about the imaginary store XYZ using a Barnes and Noble Web Service, many details are left unanswered. How is store XYZ going to build its Web Service consuming application? What exactly will the XYZ's website ask of the Barnes and Noble service? How will it know what arguments to pass and in what form will it receive the result? How will Barnes and Noble make sure that it does not accept any transaction from an unauthorized website? Most of these questions will be answered when the details of the various building blocks of Web Services, like XML, SOAP, WSDL and UDDI are presented.

Some of the other real applications of Web Services are summarized in table 5-1:

Case Studies Table

Table 5-1: Case Studies

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Allegis Corp.	Develops and markets collaborative commerce solutions that help multiple companies sell and market together	Customers use a variety of software architectures	Allegis makes its collaborative solutions available over Web Service so customers can access these services regardless of their internal software system.	Microsoft
CenterPost	Provides solutions that automatically deliver interactive messages over any communications channel – voice, wireless, email, fax and instant magazine	The partners of the company use variety of software architectures.	SOAP messages are used to access CenterPost's database via a SOAP listener and a COM+ component.	Microsoft
CertifiedMail.com	CertifiedMail.com offers secure messaging solutions to a wide range of customers, allowing secure messaging capabilities to be easily added to a customer's existing system	To easily interoperate with the various software architectures that the partners use.		Microsoft

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Continental Airlines	The fifth-largest airline in the U.S.	Unnecessary duplication of data for different user interfaces	Web Services are used between the actual data to be displayed and the different types of user interfaces (PC browser, handheld devices) to minimize unnecessary duplication.	Microsoft
Australian Bureau of Statistics	Provides statistical information about virtually every field to a variety of commercial and Governmental projects	Inefficiencies within bureau departments	Web Services are used to streamline the workings of various departments of the bureau.	IBM Application Briefs
Centrala StudieStödsnämnden (CSN)	Swedish government's banking authority responsible for providing financial aid to students	The existing infrastructure of the company needed improvement	Web Services are used to provide data to the Interactive Voice Response system and to the Web Site of the CSN.	IBM Application Briefs
METRO AG	METRO Group is a capital market-orientated, highly-competitive retail group with an international profile. It is the fifth largest trading group in the world	Interoperability needed for the "smart-shelf" technology used in the retail stores of the company that have components that are based on different software platforms (like Win2K, Linux, etc.).	The Web Service acts as intermediary between the components of the 'smart-shelf' system.	IBM jStart Case Study
J.D. Edwards and company	One of the world's leading developers of agile software solutions.	The company is exploring Web Service so as to easily inter-operate with the customers existing IT infrastructure.	The company wants to expose its own services as Web Service and also use third party applications as Web Service. The company expects to use both public UDDI registries and private UDDI registries in publishing and discovering of Web Services.	IBM jStart Case Study

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Integrated Shipbuilding Environment Consortium (ISEC)	The ISEC is a joint Industry/Government initiative. The focus of this initiative is on developing best of breed information technology, communication and management capabilities to facilitate collaboration amongst Shipbuilders, suppliers and manufacturing companies.	The ISEC wants to streamline the purchasing practices of its members by bringing uniformity across the systems. For that, interoperability between various infrastructures is necessary.	ISEC defined a UDDI in which all suppliers put in their product information. The shipyards customers get the information from the UDDI and then buy the required merchandise directly using the supplier's system through Web Service.	IBM jStart Case Study
Ford Motor Company	The world's second largest automaker	The tightly integrated old components of the company's IT systems had to be replaced.	Ford's intent is to work with IBM to develop a partial reference implementation of the architecture as a proof of concept using WebSphere and Tivoli products, and to migrate an existing engineering management application as a proof point. The architecture will ultimately embrace requirements, such as publishing and retrieval of service metadata from UDDI registries, and common services for security and Web services management.	IBM jStart Case Study

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Elsevier Science	Undisputed market leader in the publication and dissemination of literature covering the broad spectrum of scientific endeavors	Tight coupling between various components was causing the company to write a lot module-to-module, platform specific code to make its product useable for all sorts of customers.	The company's product will consist of loosely coupled independent Web Services.	IBM jStart Case Study
Deutscher Stadte- und Gemeindebund	Represents the autonomous administration of communities of town and cities within Germany and Europe.	The various German communities use a variety of IT systems hence electronic integration and digitized business processes were difficult to establish.	The German government will most likely host a number of different UDDI registries. An eGovernment Service Centre will be responsible to build a prototype, determine additional standardization, provide administrative and production services for such a registry.	IBM jStart Case Study
Credit Union Electronic Transaction Services (CUETS)	A Canadian organization that provides a complete range of advanced payment systems. A member of MasterCard International	The company wants the ability to connect any client, partner or supplier to any of its services regardless of the IT systems involved.	A Web Service acts as middle layer between the client presentation layer and the database layer.	IBM jStart Case Study
Cranfield University, Silsoe	Cranfield University is one of Western Europe's largest academic centers for strategic and applied research, development and design.	NSRI (National Soil Resources Institute), a center within the University wants its users to be able to access the data they need in the format that they need it.	University Institutes, Centers and Departments can register themselves with the UDDI registry, identify what data holdings they wish to make available, and the location(s) of such data holdings. Users can then access the registry and see the data that is available.	IBM jStart Case Study

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Con-Way Transportation Services, Inc.	A $2 billion transportation and services company	The company wanted to replace the labor intensive and error-prone paper-based documents used for transmitting invoices but the expensive Electronic Data Exchange (EDI) was not an option for its small and middle-sized customers.	Web Services option was explored.	IBM jStart Case Study
Bonndata GmbH	Bonndata is the IT provider for 'Deutsche Herold', a well established finance and insurance corporation in Germany.	The company had to transfer their data to its partners in bulk via an ISDN line once a week. A more functional way of achieving this objective was needed.	The Web Service lets the partners of the company download the data on demand either directly or through HTTP based communication supporting bulk download as well as individual queries.	IBM jStart Case Study
Baltimore Technologies	Baltimore Technologies' products, services and solutions solve the fundamental security and trust needs of e-business.	The customers of the company had a problem of doing further processing of the data provided by the company leading to difficult-to-track errors in the use of the data. Also the customers did not have flexible and on-demand access to this data.	The COM component in customer's web servers can communicate with the company's Web Service security using the SOAP digital signature. The decoupling brought about by SOAP-XML between the customers' IT systems implementation with that of the company also makes it easy for the company to add new customers.	IBM jStart Case Study

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Street Line	A member of Credit Suisse, an affiliate of Swiss American Securities Inc., a broker-dealer and cross-border custodian.	The company runs a third party application on its server to add features for its users. Hosting an application causes problems like server resource utilization, bug-fixes and updates, maintenance problems, etc. Using this application as a Web Service was a better option.	As and when the features of the third party application are needed the company's server sends a request to the third party Web Service and the third party application responds.	IBM jStart Case Study
TCI	Training & Consulting International AG (TCI) is one of the top ten job training and placement organizations in Germany. Under contract to the German Federal Department of Labor, TCI provides a range of services designed to facilitate job placement and fulfillment.	The company wanted to streamline its operations by using B2B integration, but it had to overcome system barriers like legacy systems to PC systems to voice and multimedia integration.	Web Service was used to interact with the database layer of the TCI systems. Security was achieved by using SSL for transport and SOAP message encryption.	IBM Application Briefs
RTS	Ricoh Technosystems (RTS) is a ¥110 billion-a-year enterprise based in Japan with 6,300 employees. RTS offers one-stop support and service for the design, deployment and maintenance of business/office solutions.	The company needed a way to integrate with the services provided by various educational service companies.	The services provided by the education service companies were published in UDDI public directory and the RTS Web Service accesses these services.	IBM Application Briefs
Store	Norway 's largest provider of	The company wanted to replace	A COM object in the employers'	IBM

Organization Name	Overview of Company	Why Web Services were used	How Web Services were used	Source
Brand	pension plans, insurance and other financial services,	the manual method of updating its database with its plan-holders' salary details with automatic processes.	payroll application synchronizes the changes in employees salary details with the company's database by sending SOAP messages to the company's Web Service.	Application Briefs
Advanced Technologies Systems	Advanced Technology Systems (ATS) designs and builds e-business systems for government and commercial clients.	Public safety officials wanted to build a communications infrastructure that will let officers in the field access criminal justice information over a secure private network via a Web browser.	A device carried by the officer sends vehicle data entered by the officer as SOAP message which is converted into legacy transaction and sent to the crime database for querying. The data from the database is sent back as legacy transaction and again converted back into SOAP message and transmitted to the officer in field.	IBM jStart Case Studies
ABN AMRO Bank	The Electronic Support Services (ESS) group in ABN AMRO's Chicago office hosts a series of web sites for a variety of industries	The company wants to offer granular services to other organizations' websites. To interoperate with their technologies the company needs to use Web Service.	SOAP based Web Service is used to deliver payment services which are basically HTML pages sent via HTTPS protocol.	IBM jStart Case Studies
AstraZeneca	AstraZeneca is one of the world's leading pharmaceutical companies.	The company needed a service so that its various applications access the LDAP directory of its employees. But at the same time it wanted to be flexible enough to allow for future changes in the LDAP schema.		IBM jStart Case Studies

References

Cantara M. (2003 June), *Web Services Pervade Systems Integration Projects*. Gartner Research.

Giantsopoulos, Tom (2002 April), *Common Picture eXchange Environment (CPXe).* PowerPoint Presentation, 21 slides
Retrieved from http://www.i3a.org/powerpoint/CPXe_Presentation.PPT

Microsoft, *XML Web Services Case Studies*,
Retrieved from http://www.microsoft.com/net/business/casestudies.asp

Smith D. and D. Heuter (2003 June), *Digital Photo Opportunities Develop via Web Services*, Gartner Research.

6. Supply Chain Architecture:

Introduction

A supply chain is a collection of all the links involved in converting raw materials into a usable product and delivering it to its intended consumer. Every industry has a specialized supply chain architecture specific to the industry, e.g. an automotive supply chain or an electronics product supply chain, and no two supply chains are identical. Even within a single industry, one company's supply chain is going to have a different architecture than another. This chapter tries to point out some of the common aspects that are present in all supply chains with the help of the current literature on the subject.

Supply Chain

A supply chain fundamentally consists of everything that is involved in delivering a product or service to end-users. The concept of a supply chain has evolved from a simple structure in the old industrial age to its current, complex form (Hugos, 2003).

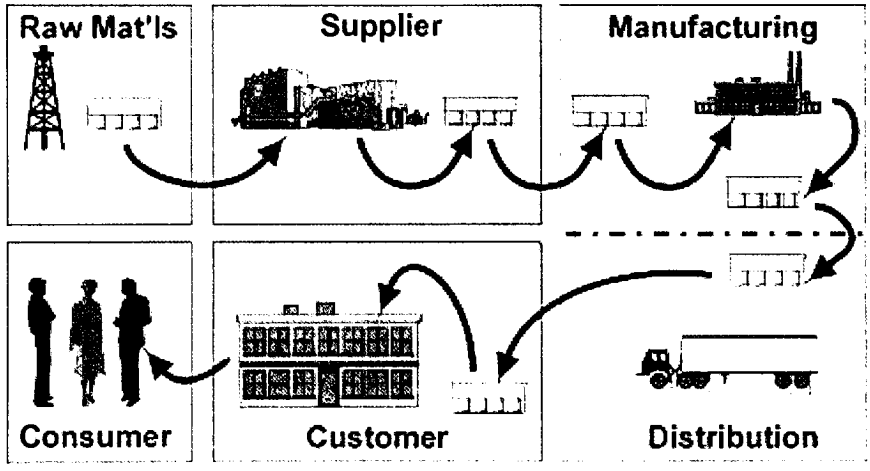

Figure 6-1: Basic Elements of a supply chain

In the 1900s, companies were vertically integrated, i.e. a single organization would own most, if not all, steps in the supply chain: manufacturing of raw materials, transportation services, product manufacturing, distribution and retail showrooms. This gave organizations the benefit of large scale production, but the major disadvantage of inflexibility to product variation. This was acceptable in the old industrial economy, but as consumers started demanding variety in products and demands began to fluctuate, this structure was no longer feasible. To keep up with globalization and rapid technological changes, the supply chain has split into a multitude of companies, each performing a single activity in the supply chain.

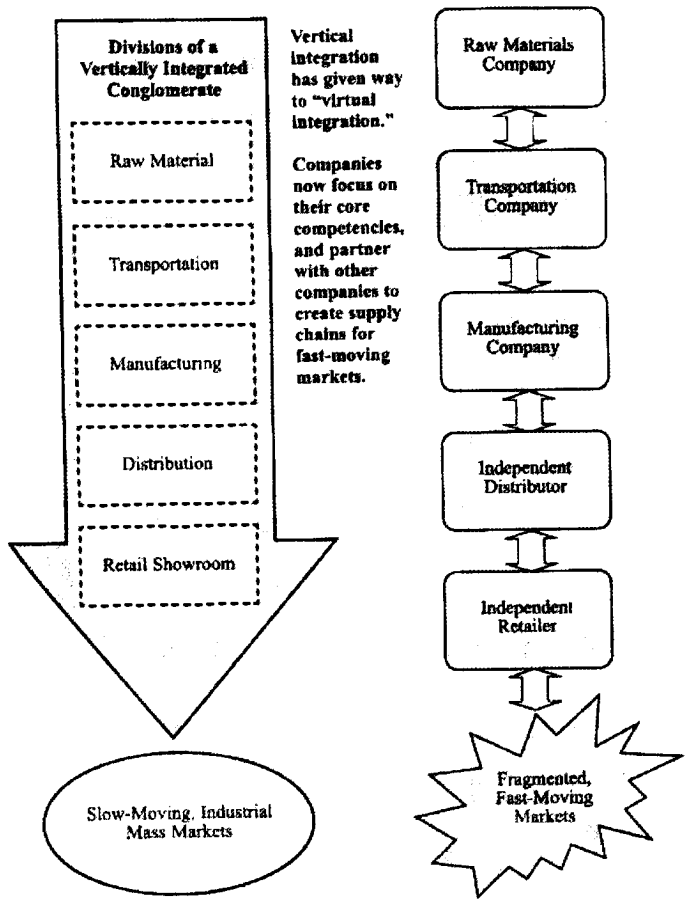

Figure 6-2: Evolution from a vertically integrated supply chain to a "virtually integrated" supply chain (Hugos, 2003)

The modern concept of a supply chain can be defined in many different ways. The definition can include a number of factors outside of a single organization's control or it can be narrow, depending on the perspective of the definer (Ayers, 2002). A few definitions of a supply chain follow:

"A supply chain is the alignment of firms that bring products or services to market" (Lambert et. al., 1998)

"A supply chain consists of all stages involved, directly or indirectly, in fulfilling a customer request. The supply chain not only includes the manufacturer and suppliers, but also transporters, warehouses, retailers and customers themselves." (Chopra and Meindl, 2001)

"A supply chain is a network of facilities and distribution options that performs the functions of procurement of materials, transformation of these materials into intermediate and finished products and the distribution of these finished products to customers." (Ganeshan and Harrison, 1995)

"Life cycle processes supporting physical, information, financial, and knowledge flows for moving products and services from suppliers to end-users." (Ayers, 2002)"

"Extended Supply Chain refers to the integrated set of activities completed by the full supply chain participants (suppliers, manufacturers, distributors, retailers/customers and consumers/end-users. It effectively includes the supply chain activities of each player in the channel." (Copachino, 1997)

Traditionally, organizations were focused on improving their own performance without considering the impact of their business partners' performance. But as can be seen from the definitions given above, a supply chain's success depends on all of the members involved in the product life, from raw material to recycling. In the modern marketplace, to remain in business, organizations have to further improve their performance by effectively collaborating with other companies in the supply chain of its product or service. This is made possible by innovations like the Internet and Internet based technologies like Web Services. An organization has to work with the other members in the supply chain to make sure that its product or service is available at the right place, at the right time and in the right amount. This has made the field of Supply Chain Management a hot topic of research.

In its simplest form, a supply chain consists of a company, its suppliers and its customers. An extended supply chain will additionally have a supplier's supplier (or ultimate supplier) and, similarly, an ultimate customer. Additionally, there is a whole category of companies that are solution providers to other companies in the supply chain. These are companies that provide services in logistics, finance, marketing and information technology (Hugos, 2003).

A typical supply chain has the following participants:

Producers

Producers or manufacturers are companies that make a product. The product can be raw materials or finished goods. Subsequently, there will generally be more than one producer in a supply chain. A producer can also create products like music, software, a design or a service that may not need any raw materials in the traditional sense.

Distributors

Distributors or wholesalers are intermediaries between manufacturers and customers. Distributors take inventory in bulk from producers and generally sell to other businesses: the retailers. Distributors perform the functions of product promotion, sales, tracking customer needs, inventory management, warehouse operations and product transportation.

Retailers

Retailers stock inventory and sell directly to retail consumers. This organization closely tracks the preferences and demands of its customers. It advertises to its customers and tries to attract them using seasonal promotions, low prices and/or product selection, services, convenience, etc.

Customers

A customer can be an individual who buys a product to consume it, or an organization that purchases the product in order to incorporate it into another product that it will, in turn, sell to other customers.

Solution Providers

Solution providers or service providers are organizations that provide services to producers, distributors, retailers and customers. Service providers have the necessary expertise to perform certain functions that are needed in a supply chain of any kind of product or service. Instead of each producer investing in managing such functions by itself, solution providers can do it more effectively and economically. Common service providers are providers of transportation and warehousing. These are trucking companies and warehouse companies. Apart from such logistics providers, there are financial service providers like banks, credit rating agencies and collection agencies that can make loans, perform credit analysis and collect past dues, respectively. Other service providers provide information technology and data collection services. There can be specialized service providers within a particular type of industry which provide services that cater specifically to that industry's needs, e.g. the health-care industry, which is heavily regulated by the FDA, will need a specialized service provider to take care of FDA compliance.

As mentioned before, no two supply chains are the same; hence it is not possible to draw a structure that can fit all possibilities. The figures 6-3 to 6-5 show how a basic supply chain can be constructed.

Figure 6-3: A basic supply chain (Hugos, 2003)

The figure 6-3 shows the simple structure that is fundamental to any supply chain. However, it is far from representing the true, complex nature of supply chain architecture.

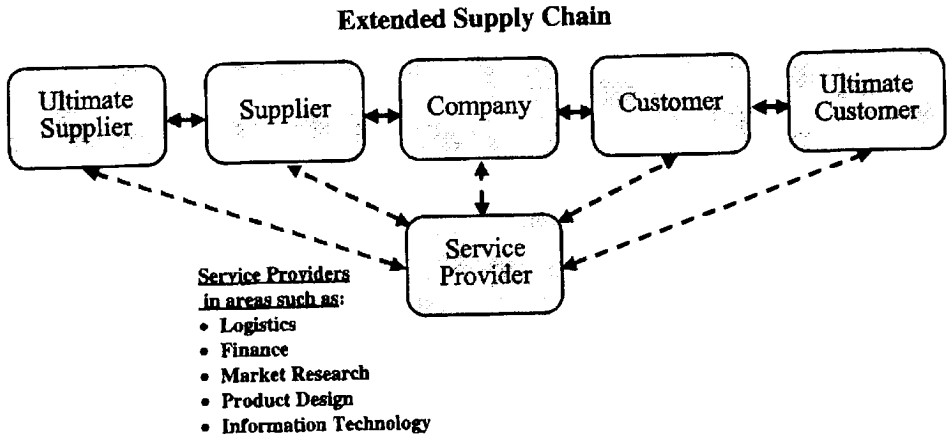

Figure 6-4: A complex supply chain (Hugos, 2003)

This shows a more realistic supply chain and can be considered to be the most generic form. Most of the supply chains can be fit to this pattern.

Figure 6-5 is an example of an extended supply chain that includes specific service providers.

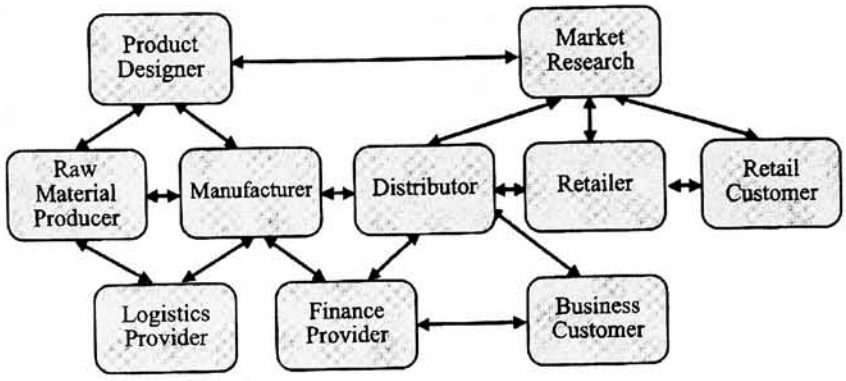

Figure 6-5: An example of a supply chain (Hugos, 2003)

As seen in figure 6-5, a typical extended supply chain example can be constructed by including, a producer, a supplier and a customer as the most basic participants, different types of additional customers and suppliers, auxiliary manufacturing units and any logical combination of the solution providers.

Following are some real-life supply chain models found in the literature.

Deere & Co (Sheridan, 1999)

Deere & Co's Worldwide Commercial & Consumer Equipment Division (WC&CED) produces small tractors and commercial mowing equipment. Most of the production, as well as logistics and other non-core functions, have been outsourced. The company has a large supplier base, some of which provide raw materials for manufacturing while others provide readymade parts which can be directly used in assembly.

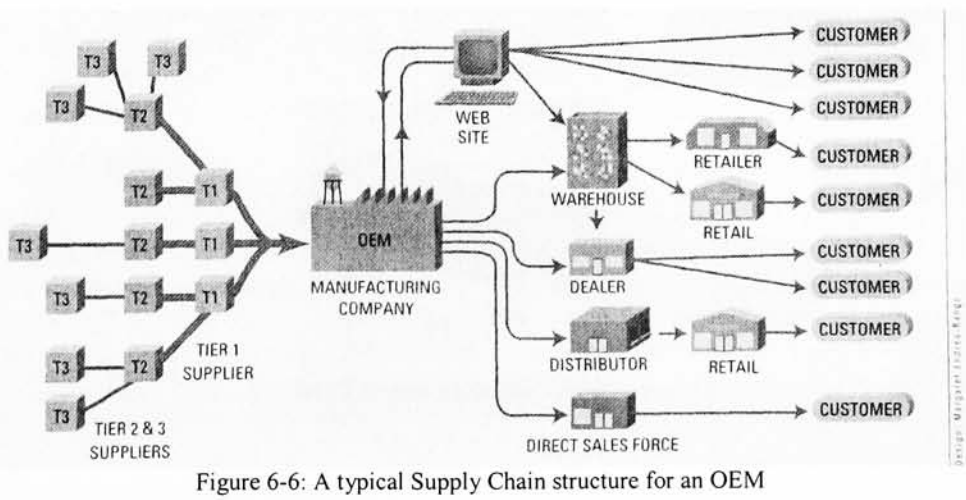

Figure 6-6: A typical Supply Chain structure for an OEM

The customers for WC&CED are dealers, which then sell directly to end users. In the past, WC&CED would directly push its products out to dealers. This caused them to

stock an entire year's worth of lawn and garden equipment. This is an expensive inventory, which was financed by Deere. Instead, to reduce this cost and to have more flexibility in responding to end user demands, WC&CED built a central warehouse, where a moderate level of finished goods inventory would be maintained to ensure fast and reliable delivery of products to replace those that dealers had sold. This way, instead of having an inventory at multiple dealer locations, the buffer inventory is stored only at a central warehouse. The figure 6-6 shows what a supply chain structure for a typical OEM (Original Equipment Manufacturer) looks like. As seen in figure 6-6, the company has Tier 1 suppliers, Tier 2 suppliers and so on. Similarly, on the other end of supply chain, the end-user may buy the products directly from the manufacturer via a website, or there can be a warehouse, distributors and then retail sellers in between the manufacturer and the customer.

Verner Frang AB (Kogg, 2003)

Verner Frang AB is a small Swedish textile trading company. In the initial stages, the supply chain structure of their business was much simpler with the company operating as an agent between the spinneries in Peru and the customers in the European market. As the company grew, it increased its product range and the figure 6-7 shows how their current supply chain is structured. The company has suppliers at multiple tiers. It interacts directly with the suppliers at each tier as shown in the figure 6-7. The company's unit in Peru works with a certifier who checks for the environmental friendliness of the products. The figure 6-7 does not show the customers of Verner Frang, but does show a significant portion of their supply chain.

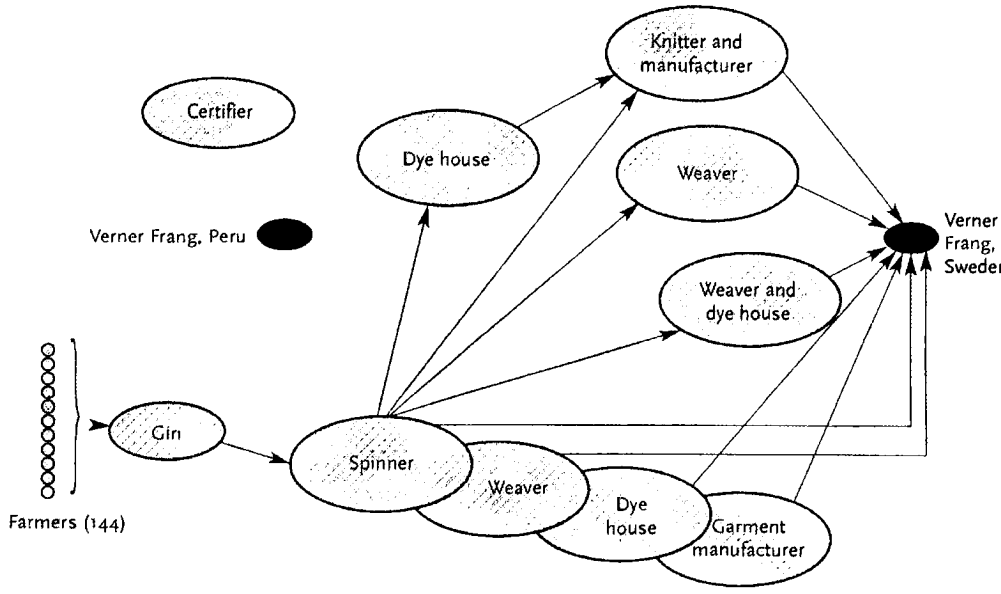

Figure 6-7: Supply Chain for Verner Frang, AB

WN, Wine Producer (Hoek, 1997)

WN is a European wine producer which acquires base wines from suppliers in Australia, South Africa, South America and southern and eastern Europe. WN

processes and blends these wines, finishes them with additives and then bottles them. Bottled wines are then labeled, packaged and shipped off to European customers. Most of these customers are wholesalers and large retail chains. The figure 6-8 shows this chain in a graphical manner.

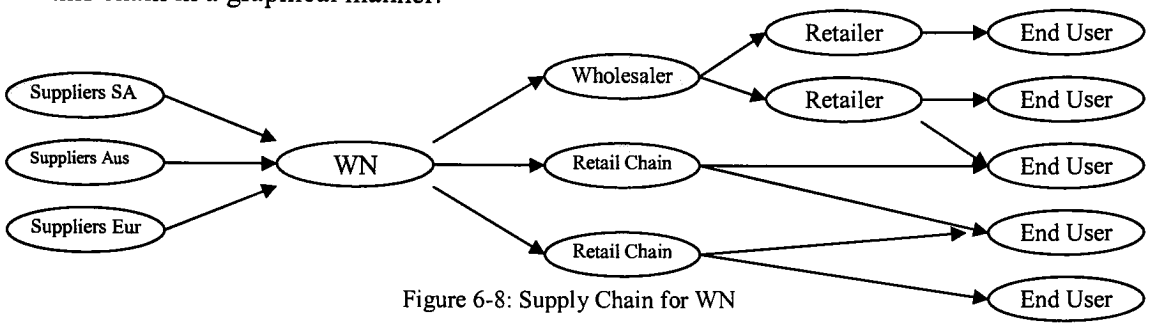

Figure 6-8: Supply Chain for WN

Whirlpool Corporation, Stanley Engineered Components (SEC) and their suppliers are manufacturing firms in the household cooking range industry. At the bottom of this supply chain are the suppliers that provide a wide range of raw material to SEC. The raw materials include rolls of steel, springs, electric motors, stamping dies and assemblies. SEC is a major supplier of products to the appliance industry. Their main products are oven door latching mechanisms and oven door hinges. Along with Whirlpool, SEC has other customers like GE, Frigidaire, Amana and Maytag. The range division of Whirlpool Corporation manufactures and markets freestanding gas and electric cooking ranges. Whirlpool markets these ranges to the end user via retail stores and chains. The figure 6-9 shows the supply chain in graphical manner.

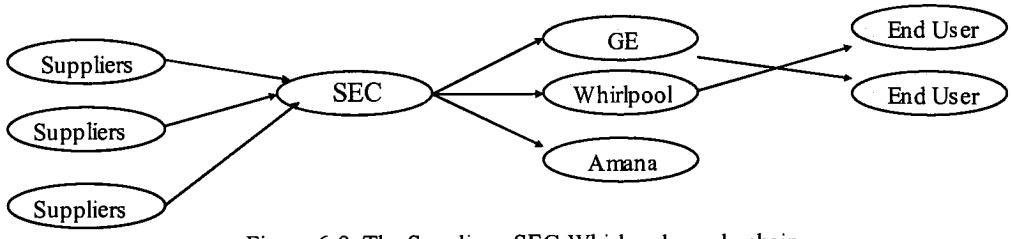

Figure 6-9: The Suppliers: SEC-Whirlpool supply chain

Dell's supply chain management is far more aggressive than the companies involved in previous case studies. Dell manufactures more than 50,000 computers every day, but carries only four days worth of inventory. Dell has about 250 suppliers responsible for delivering over 3500 components. Dell's suppliers maintain about 8-10 days of inventory. Any high inventory will give rise to obsolescence and high costs of

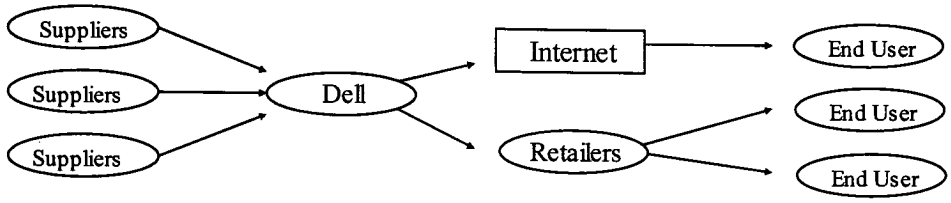

Figure 6-10: Dell supply chain

carrying inventory. But low inventory will create problems of backlog. Dell is constantly identifying, gathering and sharing new types and levels of data to manage inventory levels.

Dell receives about half of its daily orders through the internet. These orders are recorded by the company's legacy management system and then released to manufacturing. Dell creates a production schedule based on these orders and sends a message to third party logistics providers, supplier logistics centers and hubs for the required parts. The hubs have 90 minutes to pull the material out of their inventory and deliver it to Dell. Dell ensures constant fulfillment of demand by making sure that the demand for the part which is expected to be in short supply is reduced. Dell achieves this by temporary promotions for replacement parts on its website. e.g. if a 60 GB hard-drive is not available, Dell will offer an 80 GB hard-drive for the same price as a 60 GB hard-drive. Dell can offer such promotions on its website within a couple of hours of recognizing a demand/supply problem. Dell also makes sure that it has more than one supplier for each of the components it needs (except processors, which come from Intel Corp. exclusively).
Based on the information about supply chain structures mentioned above, the model shown in figure 6-11 was synthesized as a representative supply chain for illustrative purposes as well as to present this applied research project.

Representative Supply Chain

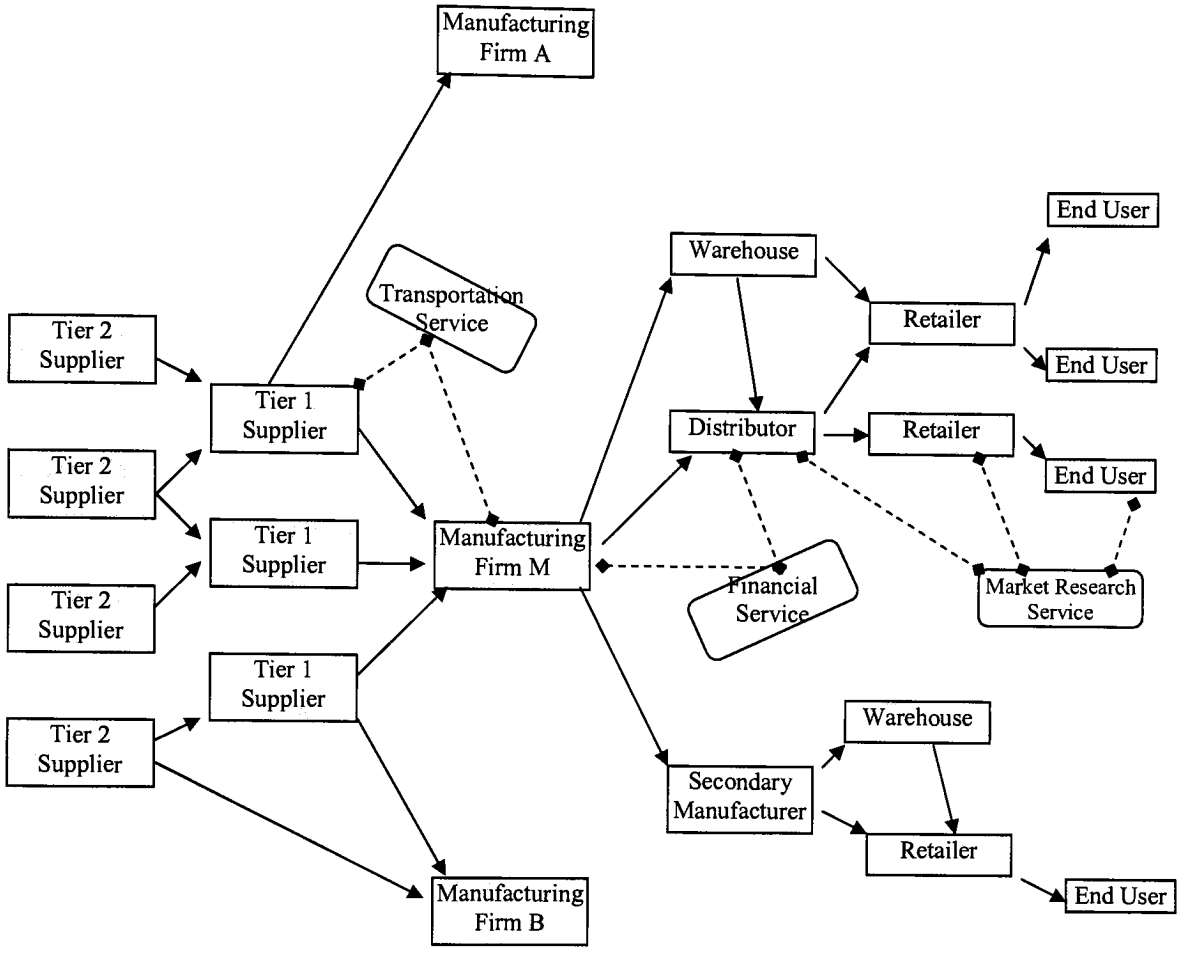

Figure 6-11 Representative Supply Chain

The diagram shows manufacturing firm M having multi-tier suppliers. For an extended view of the supply chain, it is necessary to realize that each one of the suppliers is going to have additional customers other than the firm M, which will have an effect on the supplier's responsiveness to the varying demands of firm M. The manufacturing firm M has customers in the form of wholesale distributors and secondary manufacturers. Secondary manufacturers purchase supplier M's product and use it in their own product. Firm M also has a warehouse for some storage of inventory that is needed for quick response to demand. Distributors further provide goods to Retailers. Retailers can also buy directly from firm M's warehouse. Retailers sell the goods directly to end users.

For every link between suppliers and manufacturers, manufacturers and warehouse, manufacturer and distributors, etc., transportation is needed to transfer parts. In some cases, the supplier or the manufacturer may be vertically integrated so that it has its own transportation functions. But in most cases, this transportation is outsourced to third party logistics providers, as shown in figure 6-11. Also, for any inter-

organization transactions, financial services are needed to check credit worthiness, debt collections, etc. The diagram figure 6-11 shows a financial service provider between firm M and its distributor. Market research firms provide services to predict consumer trends to the retailers and distributors, so that they can appropriately stock product.

The example shown in figure 6-11 is a representative model which will be further used in exploring the usefulness of Web Services in the supply chain area. As we can note, this supply chain model can incorporate the supply chain structures of various companies which are mentioned in the case studies above.

Comparing this model to the supply chain of the typical OEM as mentioned in the Deere & Co case study, we can see that the supply chain has multi-tier suppliers to the manufacturing firm, a warehouse, a distributor/dealer and retailers and, finally, the end customers, as shown in the representative model.

For the Verner Frang AB supply chain, Verner Frang AB itself is a distributor. The Knitter, Weaver, Garment Manufacturer units are at the same position in supply chain whereas manufacturing firm M is in the representative supply chain. There are multi-tier suppliers starting from the farmers and then Ginners and Spinners. The Peru branch of Verner Frang AB uses the services of an independent Certifier to certify whether the products provided by the suppliers meet environmental criteria. This is an example of an external service provider.

For WN, the wine producer, there are many base wine suppliers from three regions. WN itself corresponds to the manufacturing firm M. WN has wholesalers and retails chains as customers. The wholesaler corresponds to the Distributor in the representative supply chain.

In the SEC-Whirlpool supply chain, if SEC is taken to be the first tier supplier to Whirlpool (manufacturing firm M), SEC's suppliers as shown in Figure 6-9 become the second-tier suppliers. SEC has other customers GE and Amana, which correspond to manufacturing firm A and B as shown in representative model. On the other hand, if we consider SEC to be the main manufacturing firm M, then SEC's suppliers are first-tier suppliers and Whirlpool becomes secondary manufacturing firm, which makes use of SEC's products in its own products.

In Dell's supply chain, Dell is the main manufacturing firm M, with multiple first-tier suppliers. It has retailers which sell directly to end users. Dell also has its own internet infrastructure to directly sell the products to end users.

In conclusion, the representative supply chain model synthesized is truly representative, because any real-life supply chain can be fitted in this model.

References

Ayers, J. (2002), "Making Supply Chain Management Work: Design, Implementation, Partnerships, Technology, and Profits", Auerbach Publications.

Chopra, S. and P. Meindl (2001), "Supply Chain Management: Strategy, Planning and Operations", Prentice-Hall Inc.

Copacino, W. (1997), "Supply Chain Management: The Basics and Beyond", St. Lucie Press.

Ganeshan, R. and T. Harrison (1995), "An Introduction to Supply Chain Management", Department of Management Sciences and Information Systems, Penn State University.

Hoek, van, Remko (1997), "Postponed Manufacturing: a case study in the food supply chain", Supply Chain Management, vol. 2, no. 2, pg 63.

Hugos, M. (2003), "Essentials of Supply Chain Management". John Wiley and Sons, Inc.

Jacobs, D. (2003 Jun). "Anatomy of a Supply Chain", Transportation & Distribution, vol. 44, no. 6, pg 60.

Kogg, B. (2003), "Greening a Cotton-textile Supply Chain: A Case Study of the Transition towards Organic Production without a Powerful Focal Company". Greener Management International, no. 43, pg 53.

Lambert, D., J. Stock and L. Ellram. (1998) "Fundamentals of Logistics Management", Irwin/McGraw-Hill.

Mangiameli, P. C. J. Roethlein (2001), "An examination of quality performance at different levels in a connected supply chain: a preliminary case study", Integrated Manufacturing Systems, vol 12, no. 2, pg 126.

Sheridan, J. (1999 Sep 6), "Managing the Chain", Industry Week, vol. 248, no. 16, pg 50.

7. Semantic Standards

This thesis document proposes that the use of Web Services will allow multiple business units to communicate seamlessly with each other and any new business partners without investing in new technology, because all these business units will be using a standard open source technology

One major step towards achieving a free supply chain is the adoption of a suitable technology that allows free communication between business units. This document has already established that the Web Services technology is the most suitable technology for this purpose. But this alone does not solve the problem. The Web Services technology provides the means for communicating, but the content of these communication messages are not dictated by this technology. When a customer wants to send a purchase order to a supplier, there are virtually infinite formats in which it can send this purchase order. One corporation may decide to send a comma-delimited file containing the product, price, quantity, etc on a line for each material that it wants to place the purchase order for. Another corporation may decide that it will send an XML document based on a schema that the corporation itself designed. This will defeat the advantage brought forward by the Web Services technology since once again, different business partners are speaking different languages.

To better understand the need for standardization, consider an example of a Web Service that provides current temperature of a place when the service is provided with a zip code. One such service may return only the number say, 69.5 as the temperature in Fahrenheit. Another service may send back an XML document that gives the current temperature in both Fahrenheit and Celsius, e.g.

```
<xml>
 <temperature>
  <celsius>20.83</celsius>
  <fahrenheit>69.5</fahrenheit>
 </temperature>
</xml>
```

Many such possibilities exist. The developer who is using this service in his/her application will need to know this before hand so that the application can be designed accordingly and the necessary information can be extracted from this response. In the same way, the internal architecture of each business unit will need to know the format of the messages that the Web Service is going to receive and what each field means in the XML document, i.e. the semantics of the incoming message need to be known so that the application can be designed to extract the relevant parameters out of these messages. In other words, we need rigorous means of representing and conveying information over the Web that will be well suited to an audience of computers (Berners-lee, 2000).

To resolve this issue, we need to bring about standardization of the way messages are structured between communicating business parties. There are various standards that exist to describe the content encoding mechanism, examples being TCP/IP, XML. But the scenarios mentioned above indicate that standards are needed that describe the

content itself by specifying an information model. Since the intended producers and consumers of this information are computer programs, we need unambiguous definition of terms and rigorous computer-readable means of stating these definitions (Ray, 2002).

OAGIS

The Open Applications Group (OAGi) is a not-for-profit open standards group. It is an industry consortium consisting of many of most prominent stakeholders in the business software component industry, formed in 1995. This consortium has developed an XML business language known as OAGIS - Open Applications Group Integration Specification. This language consists of various Business Object Documents (BODs). Each Business Object Document is a self-describing XML document, which represents a business object like say, Purchase Order or a business action like ProcessInvoice. A business object document uses the concept of meta-data to describe itself to other software components (Connelly, 2005).

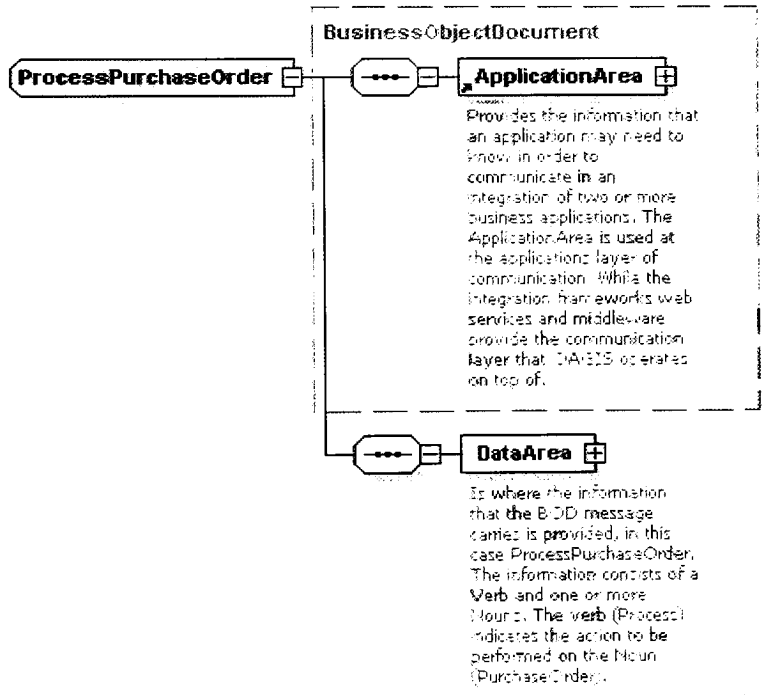

Figure 7-1: Example of a business object document (Connelly, 2005)

An OAGIS business object document consists of two components, a Business Service Request and data area. A business service request consists of a verb/noun combination, such as POST JOURNAL or GET BillofMaterial, etc. The data area has detailed information about the item represented by the noun. The BOD message architecture is independent of the transport mechanism used, whether HTTP/POST or SOAP.

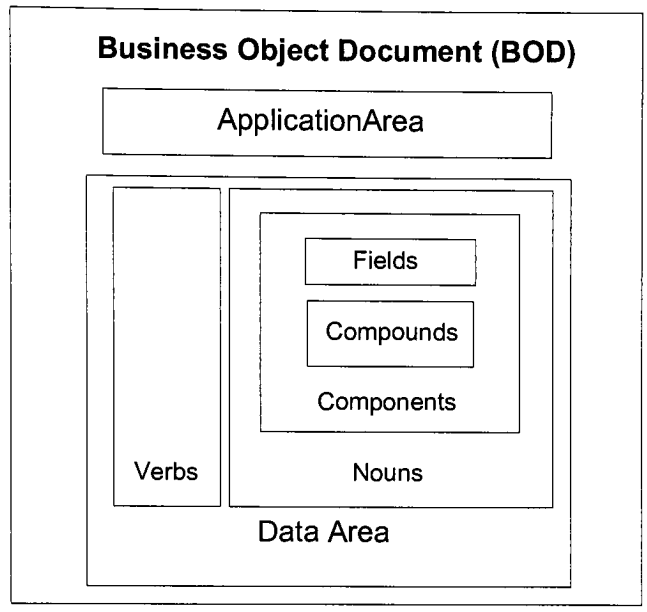

Figure 7-2: BOD Structure (OAGIS Architecture, 2005)

A BOD consists of various areas as shown in the figure 7-2.
(OAGIS Architecture, 2005).
The outermost layer of the BOD identifies the Verb, Noun, revision and runtime environment.
Application Area consists of information that can be used by the infrastructure to communicate the message.
The Data Area carries the business specific payload or data being communicated by the BOD.
Verb identifies the action being performed on the specific Noun of the BOD.
Nouns identify the business specific data that is being communicated. They are comprised of Components. Nouns can be extended according to specific needs of various vertical industries.
Components are extensible building blocks of a Noun. They are comprised of compounds and fields. Components are extensible, just like the nouns.
Compounds are basic, shared building blocks that are used by all BODs (i.e. Quantity, Amount, etc.).
Fields are the lowest level elements defined in OAGIS. Fields are fundamental elements that are used to create Compounds and Components (i.e. Description, Name, etc.).

The following shows how the metadata of a business object document looks like in the XSD format.

```
<?xml version="1.0" encoding="utf-8" ?>
- <xs:schema
    targetNamespace="http://www.openapplications.org/oagis"
    xmlns="http://www.openapplications.org/oagis"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:include schemaLocation="../Resources/Verbs/Process.xsd"
      />
    <xs:include schemaLocation="../Resources/Nouns/Invoice.xsd"
      />
  - <xs:element name="ProcessInvoice" type="ProcessInvoice">
    - <xs:annotation>
    </xs:element>
  - <xs:complexType name="ProcessInvoice">
    - <xs:complexContent>
      - <xs:extension base="BusinessObjectDocument">
        - <xs:sequence>
          - <xs:element name="DataArea"
              type="ProcessInvoiceDataArea">
            + <xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="ProcessInvoiceDataArea">
    - <xs:complexContent>
      - <xs:extension base="DataArea">
        - <xs:sequence>
            <xs:element ref="Process" />
            <xs:element ref="Invoice"
              maxOccurs="unbounded" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:schema>
```

This is the schema for a ProcessInvoice document. This defines the structure of the message when one business entity is sending a ProcessInvoice message to another entity. With the help of this metadata, computer programs can be designed that can interpret the ProcessInvoice message.

```
  <?xml version="1.0" encoding="utf-8" ?>
- <ProcessInvoice xmlns="http://www.openapplications.org/oagis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openapplications.org/oagis
    ../BODs/ProcessInvoice.xsd" revision="8.0"
    environment="Production" lang="en-US">
  - <ApplicationArea>
    - <Sender>
        <LogicalId>String</LogicalId>
        <Confirmation>0</Confirmation>
        <AuthorizationId>String</AuthorizationId>
      </Sender>
      <CreationDateTime>2001-12-17T09:30:47-
        05:00</CreationDateTime>
```

```
      <Signature qualifyingAgency="String" />
      <BODId>String</BODId>
      <UserArea />
    </ApplicationArea>
  - <DataArea>
      <Process confirm="Always" acknowledge="Always" />
    - <Invoice>
      - <Header>
        + <DocumentIds>
          <LastModificationDateTime>2002-04-
            05T00:00:00Z</LastModificationDateTime>
          <DocumentDateTime>2002-04-
            05T00:00:00Z</DocumentDateTime>
          <Description lang="en-us"
            owner="String">String</Description>
          <Note lang="en-us" author="String"
            entryDateTime="2002-04-
            05T00:00:00Z">String</Note>
          <NeedDeliveryDate>2002-04-
            05</NeedDeliveryDate>
        + <PaymentTerms>
          <UserArea />
        </Header>
      - <Line>
          <LineNumber>String</LineNumber>
        + <OrderItem>
          <OrderQuantity
            uom="NMTOKEN">3.141592653589793238462
            6433832795</OrderQuantity>
        + <UnitPrice>
          <TotalAmount
            currency="String">3.141592653589793238462
            6433832795</TotalAmount>
          <NeedDeliveryDate>2002-04-
            05</NeedDeliveryDate>
        + <OrderStatus entryDateTime="2002-04-
            05T00:00:00Z">
        + <PaymentTerms>
          <ItemQuantity
            uom="NMTOKEN">3.141592653589793238462
            6433832795</ItemQuantity>
          <UserArea />
        </Line>
      </Invoice>
    </DataArea>
  </ProcessInvoice>
```

The example shown above shows only the most commonly used fields of a BOD. A full BOD message has a lot more fields that can account for most of the needs of any manufacturing industry. Nevertheless, these business object documents are extensible to include any special needs of a particular vertical industry.

Once one such semantic standard is accepted, the proposed architecture in this thesis allows for plug-and-play interaction between business entities. It is not even necessary that an organization accept one single standard. If the marketplace has more than one popular XML standards, the organization can decide to develop their infrastructure using more than one of these standards, since the Web Services are based on XML, which is extensible. This is not possible with other technologies like CORBA, which use binary serialized objects for passing messages from one entity to another. The SOAP message sent to a Web Service can include a payload which has fields based on multiple semantic standards if needed, and the Web Service will make use of only those fields which are based on the semantic standard of its choice.

The illustrative application developed in this thesis makes use of OAGIS as the semantic standard, because it is one of the major semantic standards in the industry.

References

Berners-Lee, Tim, Fischetti, Mark, and Dertouzos, Michael L., (2000), ''Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web,'' Harperbusiness.

Connelly, D. (2005). "White Paper: Plug and Play Business Software Integration". Open Applications Group.

OAGIS Architecture. OAGIS 8.0 SP3. (n.d.) http://www.openapplications.org/downloads/oagiswsdl/oagiswsdl80.htm Open Applications Group.

Ray, S., (2002), "Interoperability Standards in the Semantic Web," ASME J. Comput. Inf. Sci. Eng., 2(1), pp. 65–69

8 Proposed Architecture

Big Picture

This architecture described in this thesis, and the illustrative application developed, deals with the communication between two separate systems as shown in figure 8-1. In a supply chain the two systems can be a manufacturer and its supplier or the manufacturer and a logistics provider or a supplier and logistics provider or any combination of the three.

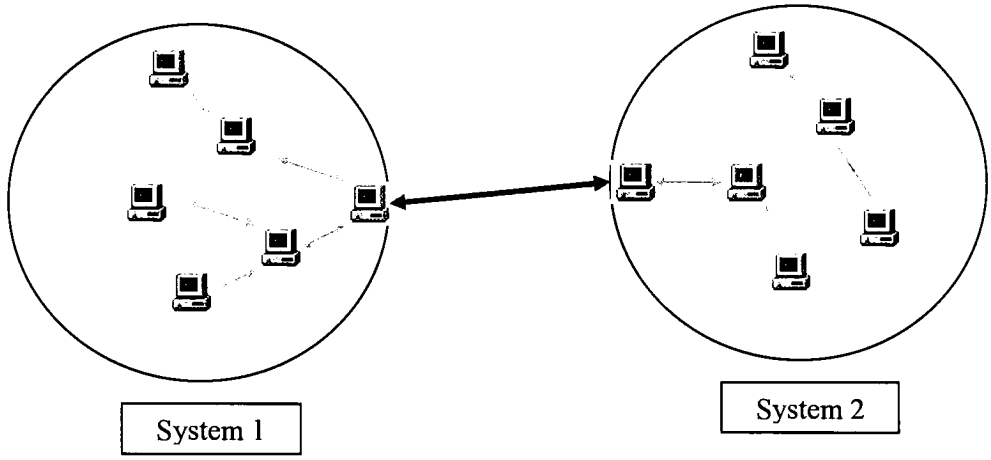

Figure 8-1: High Level Overview of Proposed Communication Architecture

As shown in the figure 8-1, we are not concerned with how the systems operate internally, but we are going to focus on how these two systems can communicate with each other.

Specifically the application created for this thesis deals with networks involving at least one manufacturer, at least one supplier and at least one carrier or logistics services provider.

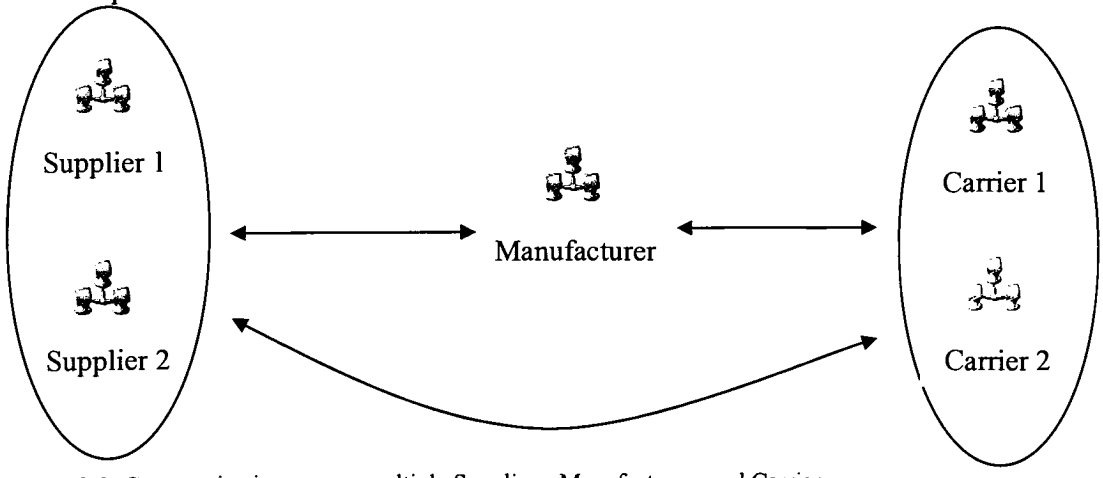

Figure 8-2: Communication among multiple Suppliers, Manufacturers and Carriers

Overview of Application

This application represents each participant in the supply chain with a User interface and a Web Service. The Web Service runs behind the screen while the user interface is tangible on the screen. A representative user interface was created for manufacturers (customers in a supply chain), another one for suppliers and one for logistics service providers. These user interfaces can accept data values from the user. The figuere 8-3 to 8-5 are the screenshots of these interfaces.

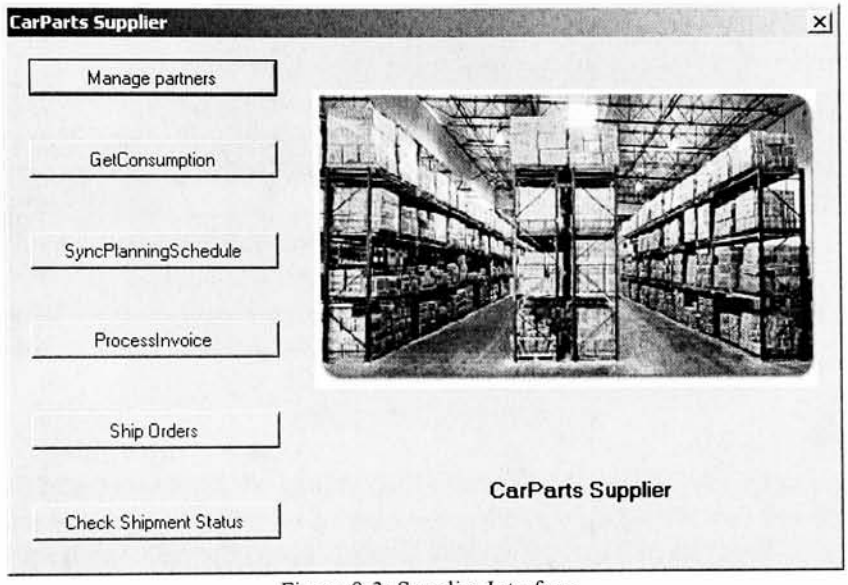

Figure 8-3: Supplier Interface

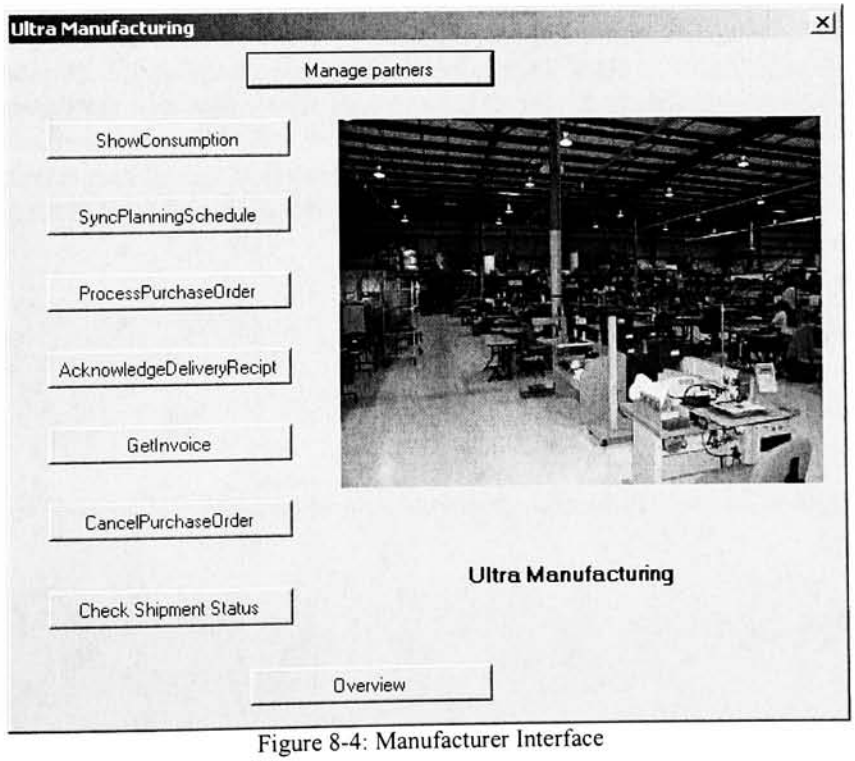

Figure 8-4: Manufacturer Interface

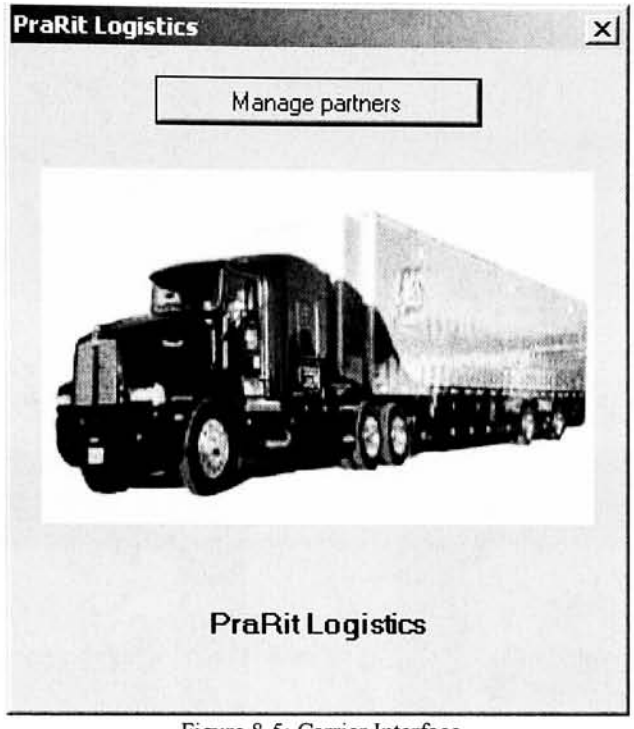

Figure 8-5: Carrier Interface

This application simulates the kind of conversation that typically occurs in a supply chain. This application is able to connect any number of suppliers with any number of manufacturers and any number of carriers. The use of OAGIS and ConWay schema enables us to simulate all possible communication scenarios like placing a purchase order, conducting a credit check, making a payment, shipment communications, etc. Only a few of these scenarios were selected for demonstration when this application was developed. The tables 8-1 to 8-7 show the specific ways in which these systems can communicate with each other, as supported by this particular implementation of the architecture proposed in this thesis. The tables show specifically what kind of information is transmitted with each message and how that message is used by the receiving party (supplier, customer or carrier).

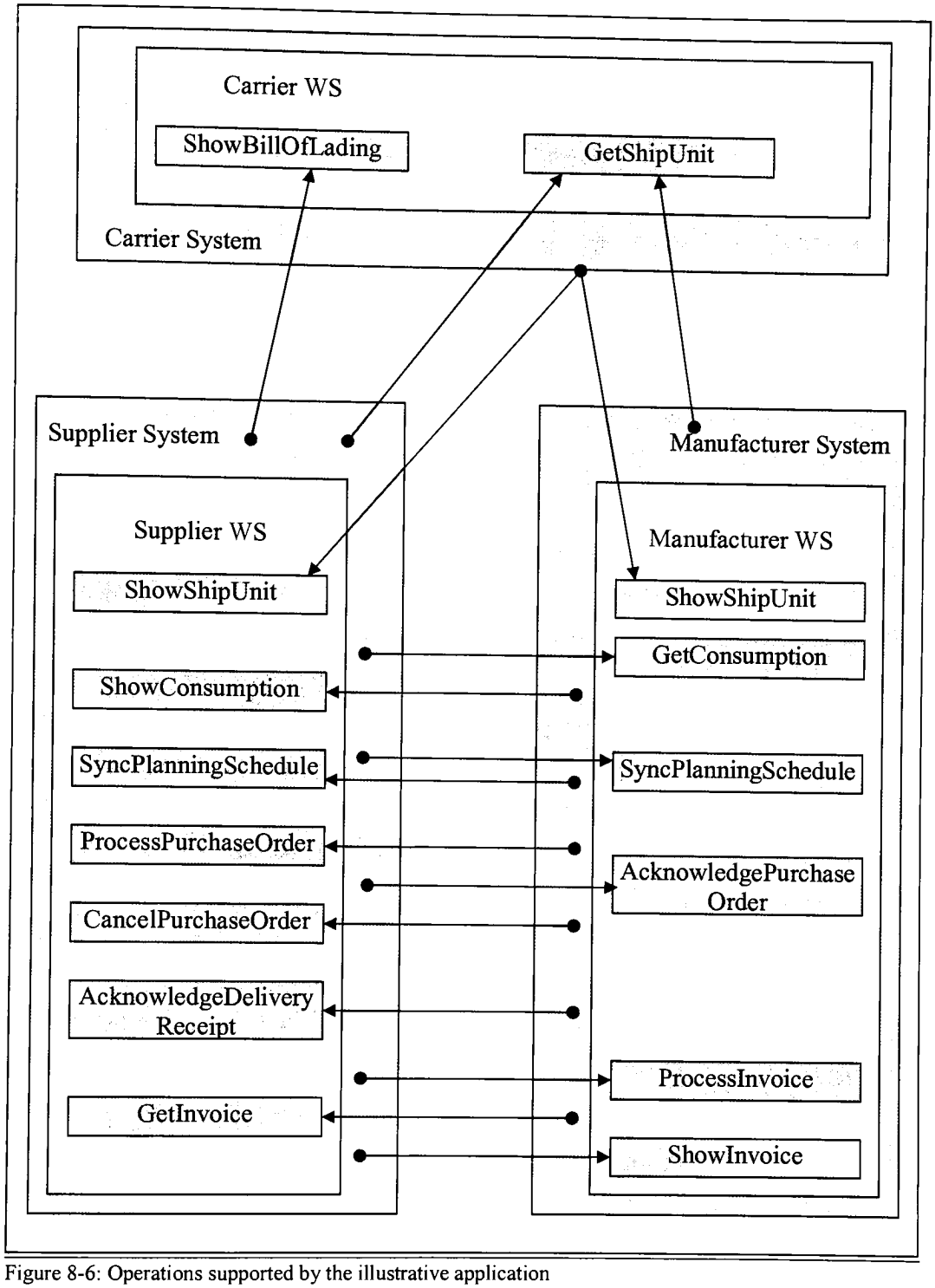

Figure 8-6: Operations supported by the illustrative application

1. Product Consumption Information:

For a supplier to be able to manage its production schedule, it is preferable to know how much of its product has been consumed by the customer who bought the product last time. Although there is no guarantee that the customer will order the same product from the same supplier again, the information will nevertheless help the supplier in its planning. The application uses GetConsumption.xsd and ShowConsumption.xsd OAGIS schema for this communication.

Table 8-1: Product Consumption Information:

No	Supplier	Manufacturer
1.1	GetConsumption: Supplier user interface sends a request to manufacturer, asking information on consumption of a particular Product at a particular location. Product = 3" tubes Consumption Location = Mfg Location 1 Manufacturer = Automanuf Corporation	
1.2		ShowConsumption: The manufacturer's system will receive the GetConsumption request. It will fill in the amount of the product consumed, store the information in its own database and send it to the supplier. Consumed Quantity = 500 Unit = tubes
1.3	Supplier will receive the consumption data from manufacturer. It will store the information in its own database and also display it as needed.	

2. Production Scheduling Negotiation:

For a supplier and its frequent customer, it is preferable to keep their production schedules in sync with each other, so that the supplier will produce the required amount of a product at the right time when the customer needs it. This will help reduce inventory for the supplier and reduce waiting time for the customer. For achieving this, the supplier and its customer will keep each other informed of their production schedules and may make modifications to these schedules to accommodate each other. The application uses OAGIS SyncPlanningSchedule schema for this.

Table 8-2: Supplier Planning Negotiation

No	Supplier	Manufacturer
2.1	SyncPlanningSchedule: Supplier will use SyncPlanningSchedule message to send information about how much of a particular product it will be able to manufacture and supply to this particular manufacturer in a specified period of time. This message can be of use to manufacturer for planning purposes, and such a message can be part of a chain of SyncPlanningSchedule sent back and forth between the supplier and the manufacturer to plan manufacturing. Manufacturer = AutoManuf Corporation Product = 3" tubes Schedule From = 5/10/05 To = 5/15/05 Quantity = 500 Unit = tubes	
2.2		The manufacturer will store in its database the information specified by the supplier and also display it as needed. The manufacturer may then choose to respond to this message independently with its own SyncPlanningSchedule message to start or continue a chain of such messages.

No	Supplier	Manufacturer
2.3		SyncPlanningSchedule: The manufacturer will use SyncPlanningSchedule message to send information about how much of a particular product it will need from a supplier in a specified period of time. This message can be of use to supplier for its own planning purposes, and such a message can be part of a chain of SyncPlanningSchedule sent back and forth between the supplier and the manufacturer to plan manufacturing. Supplier = CarParts Supplier Product = 3" tubes Schedule From = 5/10/05 To = 5/15/05 Quantity = 500 Unit = tubes
2.4	The supplier will store in its database the information specified by the manufacturer and also display it as needed. The supplier may then choose to respond to this message independently with its own SyncPlanningSchedule message to start or continue a chain of such messages.	

3. Purchase Order

When a manufacturer wants to request goods from the supplier, it will place a purchase order with the supplier. The application uses ProcessPurchaseOrder, ChangePurchaseOrder and CancelPurchaseOrder schema for this.

Table 8-3: Purchase Order communication

No	Supplier	Manufacturer
3.1		ProcessPurchaseOrder: The manufacturer will send this message to a supplier to place a purchase order. The manufacturer will send information on Product needed, quantity, Rate quote, date by which delivery is needed. It will also store this information in its database. Supplier = CarParts Supplier Product = 3" tubes Delivery Due = 5/10/05 Unit Price = 5.50 Quantity = 100 Unit = tubes
3.2	AcknowledgePurchaseOrder: The supplier will receive the purchase order from the manufacturer. The supplier will store this information in its database. The supplier can take one of following actions: Accept PurchaseOrder: Accept the PO as it is. Reject PurchaseOrder: Reject the PO. Change PurchaseOrder: Change the delivery date, Rate, and/or Quantity of the product and make a counter-offer to the manufacturer.	

3.3		The manufacturer system will display the supplier's response. Depending on the response from the supplier, manufacturer may do one of the following: Supplier accepted the PO: The manufacturer will make a note in its database that the particular PO has been accepted by the supplier. Supplier rejected the PO: The manufacturer may decide to let go of this PO, possibly to resubmit it later. Or it may change the quantity needed, needed delivery date, rate to resubmit the PO immediately to the same supplier or to a different supplier. This will count as a new PO. The original PO to original supplier will be cancelled. The manufacturer may decide to cancel the PO without submitting a new PO. In either case, canceling a PO will have no special significance, because the supplier has rejected the Purchase Order already. Supplier made a counter-offer: The manufacturer may accept the PO as suggested by the supplier. Or it can change the quantity, due date, rate or supplier and resubmit the PO. As earlier, this will count as new PO and earlier PO will be cancelled. Or the manufacturer may simply chose to cancel this PO without submitting a new one.
3.4	Depending on the response from supplier the following may happen: The manufacturer cancelled the PO: The supplier will make a note in its database accordingly. The manufacturer resubmitted the PO: Since this is a new PO, the cycle will restart from 4.2	

As shown in the figure 8-7, the application provides for modification, resubmission of the purchase order if necessary.

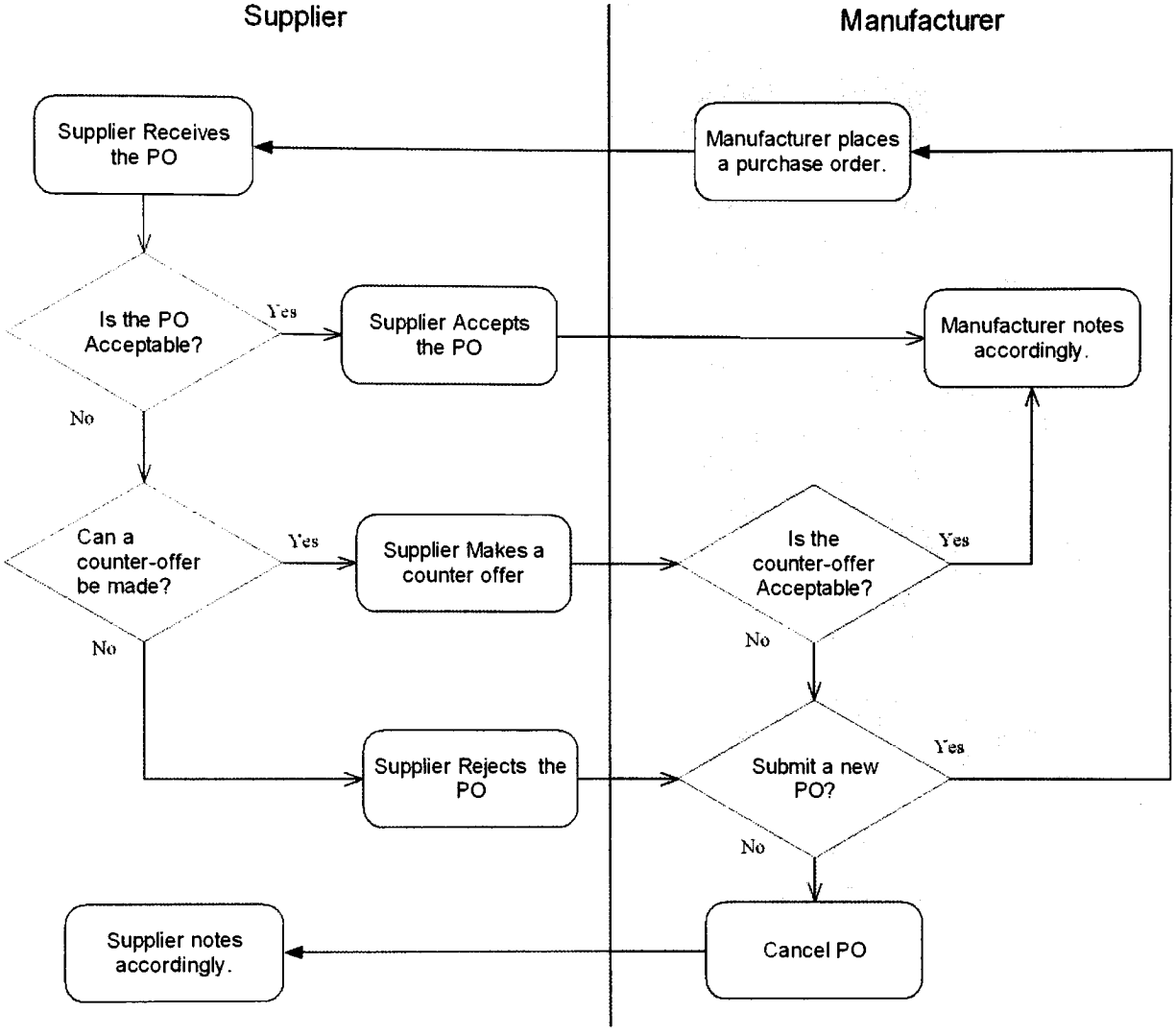

Figure 8-7: Purchase Order interactions between supplier and manufacturer

4. Shipping the orders:

When the items are to be shipped, the supplier sends a request to the carrier using the BOLRequest ConWay schema. Simultaneously, it will also send a ShowShipment message to the customer to whom the shipment is being sent, to notify it of the shipment.

Table 8-4: Shipment communication

No	Supplier	Carrier	Manufacturer
4.1	Ship Orders: When the supplier has manufactured enough product quantity to satisfy a PO, it will decide to request a carrier to transport the material to the manufacturer. It will provide information to the carrier about the total weight of the material to transport, number of pieces to transport and the type of packaging (boxes, bundles, crates, tubs, etc.), date of pick-up and will request a Bill of Lading. At the same time, supplier will send a ShowShipment notice to manufacturer letting it know that a shipment has been sent for a previously placed PO along with the name of the carrier.		
4.2		The carrier system will make a note of this transport in its database. It will calculate a simulated delivery time for purpose of illustration. It will send acknowledgment to the supplier in the form of a Bill of Lading.	The manufacturer will make a note in its database that a shipment has been sent for it.

5. Shipment Status Information for supplier:
If the supplier wants to know the status of the shipment (expected delivery date/time, etc), it will send a ConWay ShipmentStatusRequest schema to the carrier.

Table 8-5: Shipment status communication between supplier and carrier

No	Supplier	Carrier
5.1	ShipmentStatusRequest: If the supplier needs to know the status of the shipment sent to the manufacturer, it will send ShipmentStatusRequest to the carrier with the Reference number for the transport.	
5.2		ShipmentStatusResponse: The carrier will check for the Reference number in its database. If, as per the randomly simulated transport time, the shipment has been delivered, it will reply with the delivered time and status message of "Delivered". If the shipment has not yet been delivered, the carrier will respond with estimated delivery time and status message of "In Transit".

6. Shipment Status Information for Manufacturer:
Exactly same as table 8-5, but requested by the recipient of the shipment.

Table 8-6: Shipment status communication between customer and carrier

No	Manufacturer	Carrier
6.1	ShipmentStatusRequest: If the manufacturer needs to know the status of the shipment sent to it, it will send ShipmentStatusRequest to the carrier with the Reference number for the transport.	
6.2		ShipmentStatusResponse: The carrier will check for the Reference number in its database. If as per the randomly simulated transport time, if the shipment has been delivered, it will reply with the delivered time and status message of "Delivered". If the shipment has not yet been delivered, the carrier will respond with estimated delivery time and status message of "In Transit".

7. Invoice Processing:

For each Purchase Order accepted by the supplier, the customer will receive an invoice. This invoice is sent using the ProcessInvoice schema.

Table 8-7: Invoice communication

No	Supplier	Manufacturer
7.1	ProcessInvoice: The supplier will use this message to send invoice to manufacturer for a Purchase Order which has been shipped by the the supplier to the manufacturer.	
7.2		The manufacturer will store the invoice information given by the supplier in its database and display it as needed. The financial transaction between manufacturer and supplier are not covered by this system.

Implementation Details

In this architecture, each system is represented by a User Interface and a Web Service. The service runs behind the screens while the User Interface can be seen on the screen.

When we say that two systems communicate with each other, the following happens:

1. The originating system's user interface creates an XML document describing the process it wants to perform. It uses the inputs specified on the screen if needed. E.g. if a manufacturer wants to place a purchase order with a supplier, it will create a ProcessPurchaseOrder document based on the OAGIS schema. This document will look as shown in figure 8-8:

```
<?xml version="1.0" encoding="utf-8" ?>
- <ProcessPurchaseOrder xmlns="http://www.openapplications.org/oagis"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openapplications.org/oagis ../BODs/F
    revision="8.0" environment="Test" lang="en-US">
  - <DataArea>
      <Process acknowledge="Always" />
    - <PurchaseOrder>
      - <Header>
        - <DocumentIds>
          - <CustomerDocumentId>
              <Id>PO4587</Id>
            </CustomerDocumentId>
          </DocumentIds>
          <NeedDeliveryDate>2005-12-20</NeedDeliveryDate>
        </Header>
      - <Line>
        - <OrderItem>
            <Description lang="en-us" owner="String">GP275</Description>
          </OrderItem>
          <OrderQuantity uom="Unit">1</OrderQuantity>
        - <UnitPrice>
            <Amount currency="USD">15000</Amount>
          </UnitPrice>
        </Line>
      </PurchaseOrder>
    </DataArea>
  </ProcessPurchaseOrder>
```

Figure 8-8: Example payload message based on OAGIS

2. This XML document will be wrapped in a SOAP message and sent to the supplier's Web Service using the internet.

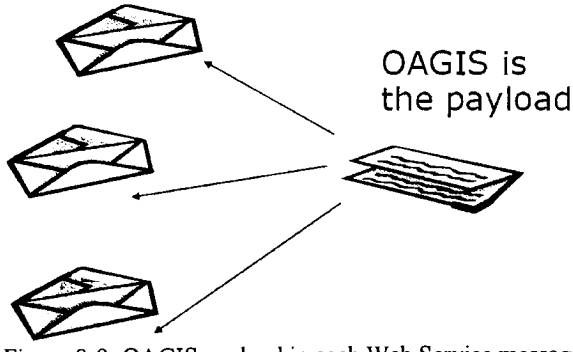

Figure 8-9: OAGIS payload in each Web Service message

3. The supplier's Web Service will receive this SOAP message and retrieve the OAGIS payload.
4. The Web Service will parse the OAGIS XML document and retrieve the relevant fields. E.g. from a ProcessPurchaseOrder document, the service will

get the Product(s) requested by the customer, Unit price quoted, quantity needed, delivery date, etc information.

In a "real life" system, once the parameters of such a request (e.g. product name, quantity, Price in case of a ProcessPurchaseOrder document) have been received by the system, the ERP system of that organization will be invoked to respond to this request. Since the application developed here is not intended to imitate the internal architecture of each system, the User Interface will instead play the role of the ERP system. When the Web Service of a system receives an OAGIS message, it will retrieve the relevant parameters from the request, as shown above, and store these parameters in an SQL table. The user interface of the system will be constantly polling the SQL database to see if any new messages are received. When the user interface detects the new message, it will query these parameters stored in the SQL table and go on to build a new OAGIS-based XML document using these parameters and user-specified inputs and respond to the originating party using the same steps as described above.

In commercial enterprises, the internal infrastructure of Manufacturer 1 will likely be completely different than the internal infrastructure of Manufacturer 2. But this application is not concerned with the internal infrastructure, but only how those systems interface with the outside world. As per the architecture described here, the internal system interfaces with the world using OAGIS schema.

- The internal infrastructure of each organization receives external requests via Web Services. These Web Services have operations whose name is the same as the names of OAGIS business objects documents (XML schema), which that operation will accept as input. This operation will respond with a confirmation message, which is the ConfirmBOD OAGIS message.
  E.g. going back to the ProcessPurchaseOrder document example, the Supplier's Web Service will have an operation called ProcessPurchaseOrder(..), which will receive a ProcessPurchaseOrder document as input.
- The business response to each request received will be asynchronous; i.e., each incoming request will be responded to with a ConfirmBOD document immediately, but actual business processing of the request will be done when the internal system is available to process the message.
- The business response to the messages will be created using OAGIS schema as well. E.g. in response to a ProcessPurchaseOrder message, an AcknowledgePurchaseOrder message will be created. This message will be sent to the originating party by calling the AcknowledgePurchaseOrder operation of the Web Service of the originating party.

Creating such an infrastructure means constructing one user interface and one Web Service for each of the systems involved in the network. The following pages describe the steps involved in creating such a system.

To create this application, you will need to install .NET Visual Studio (the C# express and Web Developer express can also be used instead of the full visual studio) and the SDK.

Converting the XML schema into directly useable C# classes:

1. Download the OAGIS schema. OAGIS 8.0 SP3 schemas were used in this program.
2. Download Con-Way schema for logistic-related operations.
3. Select the schema that you will be using in the application. E.g., To create an application that handles the Purchase Order part of the application, we would select the following OAGIS schema:
    a. ProcessPurchaseOrder.xsd
    b. AcknowledgePurchaseOrder.xsd
    c. ChangePurchaseOrder.xsd
    d. CancelPurchaseOrder.xsd
4. For the Shipment request portion of the application we will select the following Con-Way schema:
    a. BOLRequest.xsd
    b. BOLResponse.xsd
    c. ShipmentStatusRequest.xsd
    d. ShipmentStatusResponse.xsd
5. The schemas are XML documents. Before we can use these schema, we need to convert them into C# classes. This conversion of schema into C# classes can be achieved using the Xsd tool that comes with the .NET SDK. This is a stand-alone executable that can be found at <Program_Files>\<Microsoft Visual Studio 8>\bin\Xsd.exe. The conversion is carried out as follows:
    a. First we will need to combine all the schema into a single schema document. This is helpful, because this will get rid of repetition of code in multiple *.cs files when individual schema files are converted to C#. (E.g., ProcessPurchaseOrder.xsd uses the noun PurchaseOrder. When this schema is converted to C#, a class called PurchaseOrder is generated. But ChangePurchaseOrder also uses the same noun PurchaseOrder and when this schema is converted to C#, the class PurchaseOrder is re-declared in this C# file. This will result in a compilation error.)

       Combining schema files:
       If the first schema is:

```
<xs:schema xmlns="http://example.com">
  <xs:include schemaLocation="otherschema.xsd">
  <xs:element name="someelement">
   ......
  </xs:element>
```

```
</xs:schema>
```

And the second schema is:

```
<xs:schema xmlns="http://example.com">
  <xs:complexType name="someComplextype">
   ......
  </xs:complexType>
</xs:schema>
```

Then when these two schema are combined, the output should be as follows:

```
<xs:schema xmlns="http://example.com">
  <xs:include schemaLocation="otherschema.xsd">
  <xs:element name="someelement">
   ......
  </xs:element>
  <xs:complexType name="someComplextype">
   ......
  </xs:complexType>
</xs:schema>
```

Note that the xmlns attribute (at the top of the schema) for both these schema is the same value. This attribute specifies the default namespace for the objects defined in the schema, i.e., the element "someelement" and the complex type "someComplextype" belong to the namespace "http://example.com" and no other objects in this namespace will use these names. Mixing two schemas from different namespaces may be problematic, because it is possible that the two schemas use some identical name to denote two different things. That's why we will combine all OAGIS schema into one schema and all Con-Way schemas into a separate schema.

When the schema are combined make sure to adjust the paths in the <include> tags if needed.

b. After the schemas are combined we will have two schema documents, say Oagis.xsd and Conway.xsd. We will now convert the schema into C# classes. The command for doing this is as shown below:

```
> xsd   Oagis.xsd    /c    /n:OAGIS
> xsd   Conway.xsd   /c   /n:CONWAY
```

See the xsd documentation for explanation on the switches. The C# classes are automatically put into two separate namespace (OAGIS and CONWAY) for the same reasons as described above.
At the end, we will have two C# files, Oagis.cs and Conway.cs, which can now be used in the creation of Web Services as well as user interfaces. These files can be directly included in the Web Service

projects and User Interface projects as C# files or you can build a dynamically loaded library (dll) using these files and include that dll in the Web Service projects and User Interface projects. The procedure for building the dll is shown below:

```
>
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\csc.exe
/t:module   /out:schema.dll   Conway.cs    Oagis.cs
```

This will create schema.dll. Csc.exe is the C#.net compiler.

Creating a Web Service:

1. For creating and operating Web Service using .NET, you will need .NET Web Developer Express version and IIS (Internet Information Service) installed.
2. Create a new ASP.NET Web Service project in Web Developer Express and add Conway.cs and Oagis.cs in the App_Code folder of this project OR add schema.dll as a reference in the project.
3. The project will, by default, have a Service.asmx and App_Code\Service.cs file. The Service.cs file has a class called "Service", which is our Web Service. Any methods of this class with the attribute [WebMethod] before it will become an operation of the class. This file by default has following content:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo =
WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

To be able to use our schemas, we need to add two lines at the top of the file:

```
using CONWAY;
using OAGIS;
```

To create the operations supported by this Web Service, we need to create

public methods of this class with the attribute [WebMethod] attached to them.

Creating Web Service to implement OAGIS schema:

1. In this specific infrastructure, the Web Services were created in .NET using the Visual Web Developer Express. A new Web Service project was created.
2. To use the OAGIS schema, we need to include the C# classes in this project. The schema.dll (as explained above in the section "Converting schema into C# classes") needs to be included in this project as reference.
3. Each of the schemas has plenty of fields in it to describe all the details a particular transaction is performing. E.g., to describe a purchase order, a purchase order document has a Header field and multiple 'PurchaseOrderLine' fields. Each PurchaseOrderLine corresponds to a Line that a paper purchase order would have had. Each PurchaseOrderLine has plenty of fields to describe the particular item being purchased and terms and conditions of that purchase. Demonstrating the usability of all of these fields is not the objective of the thesis; hence few demonstrative fields were selected for illustration in this application.
4. Under the OAGIS-Web Service infrastructure that is being illustrated here, each Web Service will have operations that obtain a single OAGIS document as input and the name of the operations will be the same as the name of the input OAGIS document. For example, a supplier's Web Service will have an operation called ProcessPurchaseOrder, which will accept and process incoming Purchase Orders from its customers.
5. The following shows an example of how the OAGIS schema can be implemented in a supplier's Web Service, specifically the ProcessPurchaseOrder operation.

```
    [WebMethod]
    public ConfirmBOD ProcessPurchaseOrder(ProcessPurchaseOrder ppo)
    {
        //variable declarations
        //POId: Purchase Order ID specified by Customer
        //Uom: Unit of Measure
        string POId, Product, Uom;
        //Uprice: Unit Price Quoted
        //Qty: Quantity Ordered
        double UPrice, Qty;
        //delivDt: Needed delivery date by the customer
        DateTime delivDt;
        POId =
ppo.DataArea.PurchaseOrder[0].Header.DocumentIds[0].Id;
        delivDt =
DateTime.Parse(ppo.DataArea.PurchaseOrder[0].Header.NeedDeliveryDate)
;
        Product =
ppo.DataArea.PurchaseOrder[0].Line[0].OrderItem.Description[0].Value;
        Uom =
ppo.DataArea.PurchaseOrder[0].Line[0].OrderQuantity.uom;
        Qty =
(double)ppo.DataArea.PurchaseOrder[0].Line[0].OrderQuantity.Value;
        UPrice =
(double)ppo.DataArea.PurchaseOrder[0].Line[0].UnitPrice.Amount.Value;

        //logic to decide whether the purchase order should be
accepted
    }
```

This method receives a ProcessPurchaseOrder document from its customer. The supplier can retrieve the PurchaseOrder ID (POId), product description (Product), unit price (Uprice), etc information from this document as shown above. The method returns a ConfirmBod document as a reply to the manufacturer's system. What the example above does not show is what the service is going to do with the information it obtained from the input ProcessPurchaseOrder document. In a "real life" system, we will expect this information to be processed to see the credit history of the customer as well as the availability of raw material to fulfill this order by the internal system of this supplier. In this thesis, we do not make any assumption abut the internal workings of the supplier's system other than that it has the Web Service interface to the outer world as described by this infrastructure.

6. The supplier can respond to this Purchase Order by accepting it, rejecting it or modifying it to make a counteroffer for to the customer. The following shows an example of how the customer will receive this information from the supplier:

```
    [WebMethod]
    public ConfirmBOD AcknowledgePurchaseOrder(AcknowledgePurchaseOrder
apo)
    {
        string POId, Party;
        DateTime delivDueDt2;
        double UPrice2, Qty2;
        bool accepted = false, rejected = false, modified = false;

        POId = apo.DataArea.PurchaseOrder[0].Header.DocumentIds[0].Id;

        switch (apo.DataArea.Acknowledge.Code)
        {
            case AcknowledgeCode.Accepted:
                accepted = true;
                break;
            case AcknowledgeCode.Modified:
                modified = true;
                break;
            case AcknowledgeCode.Rejected:
                rejected = true;
                break;
        }

        delivDueDt2 =
DateTime.Parse(apo.DataArea.PurchaseOrder[0].Line[0].NeedDeliveryDate);
        UPrice2 =(double)
apo.DataArea.PurchaseOrder[0].Line[0].UnitPrice.Amount.Value;
        Qty2 =
(double)apo.DataArea.PurchaseOrder[0].Line[0].OrderQuantity.Value;
        Party = apo.ApplicationArea.Sender.LogicalId;
        //logic to decide whether the counter-offer is acceptable
    }
```

Here the customer retrieves Purchase Order from the AcknowledgePurchaseOrder document sent by the supplier.

9. Conclusion

This thesis has attempted to advance the art of supply chain management by studying the advances in technologies and illustrating a methodology of using these benefits, so as to make clear the practicality of these technologies in supply chain architecture. Towards that end, a communication architecture was developed for supply chain partners. The need for a semantic standard was recognized in order to successfully create such an architecture. The XML based schema developed by Open Applications groups and the ConWay Corporation were presented as possible candidates for such semantic standards. An application was developed that demonstrated the usefulness of this architecture. The application showed how the XML standards could be used for communication. and demonstrated how Web Services can be used practically in a supply chain. The importance of the UDDI standard as a registry and a connecting point between business partners was also shown.

Additionally, this thesis has developed an architecture that promises to solve the technical difficulties of communication between supply chain partners. This work shows how to develop communication networks that make use of new technologies, namely Web Services and communication standards like OAGIS. It virtually guarantees that, as these technologies and standards go into widespread use, an organization can form communication networks with another virtually as soon as they sign the business agreements with each other.

The biggest pre-requisite for success of this architecture is that all the different business organizations that can be involved in any supply chain have to agree to integrate their internal architectures to Web Services as an interface to the outer world. Furthermore, all organizations must adopt a single semantic communication standard. The big push by all major ERP vendor companies towards standardization and the previous adoption of similar standards for EDI show that this is not impossible to achieve, but can be expected to take a long time. Even if all organizations do not reach a consensus in near future, this architecture can still be used as a beneficial business model. A technology company may provide the architecture, possibly pro-bono in the beginning, and build a network of supply chain partners by offering a paid service to organizations and promising that once they join this network, they never have to bother to tweak their internal technology architectures to each new business partner they want to communicate with. This is similar to, say, a chat and messaging network like MSN or AIM, in which a newly joining user has the guarantee that he/she will, from now on, be able to communicate with millions of other users of this network.

10. Appendices

Appendix 1: Creating a Web Service in VFP:

This involves first making a COM server dll in Visual FoxPro and then creating necessary WSDL and WSML files using the VFP XML Web Service Publisher. IIS needs to be installed to create and host Web Services.

- Launch Visual FoxPro and click on File > New. Select "Project" in the file type and then click on the "New File" button.
- Save the Project in a new directory with the name WS.pjx
- In the Project Manager, on the Code tab, click on Programs and then click on "New".
- A window will show up on the screen where you can type the code of the Web Service. You can use the following code (from VFP 8 Help):

```
DEFINE CLASS ShowCustomers AS Session OLEPUBLIC
   PROCEDURE CustomersInGermany AS String
      LOCAL loXMLAdapter AS XMLAdapter
      LOCAL lcXMLCustomers AS String

      loXMLAdapter = CREATEOBJECT("XMLAdapter")

      OPEN DATABASE "C:\Program Files\Microsoft Visual FoxPro
8\Samples\Northwind\northwind.dbc"

      USE customers
      SELECT * ;
         FROM customers ;
         WHERE country LIKE "Germany%" ;
         INTO CURSOR curCustomers

      loXMLAdapter.AddTableSchema("curCustomers")
      loXMLAdapter.UTF8Encoded = .T.
      loXMLAdapter.ToXML("lcXMLCustomers")

      CLOSE DATABASES ALL

      RETURN lcXMLCustomers
   ENDPROC
ENDDEFINE
```

- In this program, a class is defined which is an OLEPUBLIC type of class. All the methods / procedures within this class will become the methods of the Web Service. The name of the class is "ShowCustomers" and the procedure defined is "CustomersInGermany". This procedure returns a string.
  Save the program in the same directory as earlier.

- Click the "Build" button to bring up the Build Options. In the Build Action, select "Multi-threaded COM server (dll)" and click OK. Save the file in the same directory as earlier with the name WS.dll
- IIS uses c:\inetpub\wwwroot as the root directory for HTML documents. Create a subdirectory in this location called WS. You have to make sure that

the account used by IIS to access this directory (see the Internet Services Manager) has the permissions to Read & Execute, Modify and Read in this directory.

- Copy the ws.dll file that you had created to this location.
- Register this COM server in the registry by using the following Run command:
  ```
  regsvr32 ws.dll
  ```
- Back in the Visual FoxPro, bring up the 'Task Pane Manager' by going to Tools > Task Pane. Click on "XML Web Services" in the toolbox near the top.
- Click on "Publish your XML Web Service". Make sure at this time that IIS is running.
- If this is the first time you are creating a Web Service through VFP, you will need to specify where you want to store the Web Service files. Here you can specify a new Virtual Location and its corresponding actual disk location or select from a Virtual Location which was already created. In the Virtual Location, type **Error! Hyperlink reference not valid.** and in the actual location, type c:\inetpub\wwwroot\WS.
- When you are done with this, a "Visual FoxPro XML Web Services Publisher" dialog box will come up asking you for the COM server. Select c:\inetpub\wwwroot\WS\ws.dll as the COM server.
- In the "Advanced" options, select ISAPI as the listener type and on the namespaces tab you can replace "tempuri.org" with say, "rit.edu".
- After exiting the Advanced Options, click on "Generate" button. This will create a WSDL file and two auxiliary WSML files which are required for linking the WSDL file to the COM server (ws.dll). These files will be stored in the location which you had specified above (c:\inetpub\wwwroot\WS).

Appendix 2: Creating a Web Service in Visual Studio .NET:

- Launch Visual Studio .NET or Web Developer Express Edition.
- Click on File > New Web Site. Select "ASP.NET Web Service" in the dialog box that appears.
- In the Location field drop down box, select "File System" and enter the directory location where you want to create the Web Service. Click OK. A Web Service has now been created.
- In the service.cs file add public methods with a [WebMethod] attribute attached on top to create operations for the Web Service. (See the HelloWord() method for example.)

Exposing the Web Service:

- Start the internet services manager by going to Control Panel > Administrative Tools.
- Right click on the default Web Site and select New > Virtual Directory.
- Type in a name for the virtual directory (it will appear in the URL for the Web Service), say "MyService".
- Set the location of the directory to the same as the directory where you created the Web Service (the directory with the file Service.asmx). Click OK.
- Edit the properties of this new virtual directory to change the ASP.NET configuration to appropriate version (1.0/1.1/2.0) if you have more than one version of .NET installed on your computer.
- Edit the directory security of your Web Service directory to allow Read and Execute access for the local IIS account (go to website > properties > directory security > anonymous access to find out the name of the account. The account name is generally like IUSER_IMERT118.) and the ASP.NET account (generally ASPNET).

Now the Web Service should be accessible at
http://<comp_name>/<virtual_dir_name>/service.asmx

Appendix 3: Creating a VFP client for a Web Service:

When you register a Web Service in the Intellisense of VFP 8, a sample code for accessing the Web Service will be automatically generated for you. You will have to add the code which will call a particular method of the Web Service.
(The following steps are not strictly necessary to write a client for Web Service.)

- Launch Visual FoxPro 8. Open the Task Pane Manager, if it is not already open, by going to Tools > Task Pane.
- Click on "XML Web Services" button near the top of the task pane.
- Click on "Register an XML Web Service" under the XML Web Services Tools.
- A "Visual FoxPro XML Web Services Registration" dialog box will show up. Here type the URL of the local file path to the valid WSDL file of Web Service. Click on "Done". In this case type, http://imert117.rit.edu/WS/ShowCustomers.wsdl (not case-sensitive).

Now you will see this Web Service in the drop-down listbox under 'Explore an XML Web Service" as "ShowCustomers". Select this service in the drop down box. Now you will see the sample code for accessing this Web Service in FoxPro. The code is reproduced below.

```
LOCAL loShowCustomers AS "XML Web Service"
* LOCAL loShowCustomers AS "MSSOAP.SoapClient30"
* Do not remove or alter following line. It is used to support
IntelliSense for your XML Web service.
*__VFPWSDef__: loShowCustomers =
http://imert117/ws/showcustomers.wsdl , ShowCustomers ,
ShowCustomersSoapPort
LOCAL loException, lcErrorMsg, loWSHandler
TRY
   loWSHandler =
NEWOBJECT("WSHandler",IIF(VERSION(2)=0,"",HOME()+"FFC\")+"_ws3clie
nt.vcx")
   loShowCustomers =
loWSHandler.SetupClient("http://imert117/ws/showcustomers.wsdl",
"ShowCustomers", "ShowCustomersSoapPort")
   * Call your XML Web service here.  ex: leResult =
loShowCustomers.SomeMethod()

CATCH TO loException
   lcErrorMsg="Error: "+TRANSFORM(loException.Errorno)+" -
"+loException.Message
   DO CASE
   CASE VARTYPE(loShowCustomers)#"O"
         * Handle SOAP error connecting to Web Service
   CASE !EMPTY(loShowCustomers.FaultCode)
         * Handle SOAP error calling method
         lcErrorMsg=lcErrorMsg+CHR(13)+loShowCustomers.Detail
   OTHERWISE
         * Handle other error
   ENDCASE
   * Use for debugging purposes
   MESSAGEBOX(lcErrorMsg)
```

```
FINALLY
ENDTRY
```

The code highlighted in **Bold** is all you need to set-up access to the Web Service. The rest of the code is related to exception handling. The code in *italics* is where the actual method will need to be called.

- After removing the un-necessary code, and adding the actual code to invoke the method on the Web Service, the complete code will look as shown below.

```
LOCAL loShowCustomers AS "XML Web Service"
LOCAL loWSHandler
LOCAL leResult as String

loWSHandler =
NEWOBJECT("WSHandler",IIF(VERSION(2)=0,"",HOME()+"FFC\")+"_ws3clie
nt.vcx")

loShowCustomers =
loWSHandler.SetupClient("http://imert117/WS/showcustomers.wsdl",
"ShowCustomers", "ShowCustomersSoapPort")

leResult = loShowCustomers.CustomersInGermany()

XMLTOCURSOR(leResult,"cur")
BROWSE
```

The **Bold** lines indicate what code was added to the earlier code to invoke the 'CustomersInGermany' method and then present the results in a suitable manner.

Appendix 4: Creating a .NET client for a Web Service:

- Open or create a Visual C# project in which you want to use a Web Service.
- Open the solution explorer view of the project, right click on the project name and click on "Add Web Reference".
- In the URL box, type in URL to the WSDL file or the '.asmx' file of the Web Service and click Go.
- Type in a name for the web reference. This name becomes the name of the namespace in which the classes created from this service are placed. Say the name is 'MyService'. Click on 'Add Reference'.
- The IDE will now create C# classes based on this WSDL.
- Now the service can be used from your code as:

```
MyService.Service mywebservice = new MyService.Service();
mywebservice.invokeoperation();
```